



Universidad Autónoma de Madrid
Escuela Politécnica Superior
Departamento de Ingeniería Informática

Deep Learning Applied to Regression, Classification and Feature Transformation Problems

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

By
David Díaz Vico
under the supervision of
José Ramón Dorronsoro Ibero

Madrid, December 17, 2021

Contents

1	Introduction	1
1.1	Machine Learning Overview	1
1.1.1	Data	2
1.1.2	Model	6
1.1.3	Training	9
1.2	Methodology	11
1.2.1	The Components of a Loss	11
1.2.1.1	Estimation	12
1.2.1.2	Approximation	12
1.2.1.3	Optimization	15
1.2.1.4	Combining the three components	15
1.2.2	Metrics	16
1.2.3	Standard Production Methodologies	18
1.3	Contributions	19
1.4	Structure	19
2	Machine Learning Core	21
2.1	Linear Models	21
2.1.1	Linear and Ridge Regression	22
2.1.2	Fisher’s Linear Discriminant Analysis	23
2.1.3	Iterative Linear Model Training	24
2.1.3.1	Linear and Ridge Regression	24
2.1.3.2	Logistic Regression	25
2.1.4	Linear Support Vector Machines	27
2.1.4.1	Linear Support Vector Classifiers	27
2.1.4.2	Linear Support Vector Regressors	29
2.2	Kernel Models	31
2.2.1	Kernel Ridge	31
2.2.2	Kernel Fisher’s Discriminant Analysis	32
2.2.3	Kernel Support Vector Machines	34
2.2.3.1	Kernel SVC	34
2.2.3.2	Kernel SVR	35
2.2.3.3	Complexity and Sparsity of Kernel SVMs	35
2.3	Deep Models	35
2.3.1	Approximation	37
2.3.1.1	The Artificial Neuron	37

2.3.1.2	Representation Learning and Deep Learning: The Multilayer Perceptron	39
2.3.1.3	Spatial data and structured features: The Convolutional Neural Network	41
2.3.1.4	Temporal data and sequential patterns: The Recurrent Neural Network	43
2.3.2	Optimization	44
2.3.2.1	Back-propagation	45
2.3.2.2	Underfit	46
2.3.2.3	Overfit	50
3	Deep Neural Network Prediction in Wind and Solar Energy	53
3.1	Summary	53
3.2	Publications	53
	Deep Neural Networks for Wind and Solar Energy Prediction	57
	Deep Neural Networks for Wind Energy Prediction	75
4	Deep Fisher Discriminant Analysis and Imbalanced Classification	89
4.1	Summary	89
4.2	Publications	89
	Deep Least Squares Fisher Discriminant Analysis	93
	Deep Fisher Discriminant Analysis	105
	Deep MLPs for Imbalanced Classification	117
5	Deep Support Vector Machines and Companion Losses	125
5.1	Summary	125
5.2	Publications	125
	Deep support vector neural networks	129
	Deep Support Vector Classification and Regression	143
	Companion Losses for Deep Neural Networks	155
6	Conclusions and further work	167
6.1	Conclusions	167
6.2	Further work	168
7	Conclusiones y trabajo futuro	171
7.1	Conclusiones	171
7.2	Trabajo futuro	172
	Bibliography	173
	A Publications	185
	B Software	189

Abstract

The decade from 2010 to 2020 has seen a series of impressive improvements in the performance of Machine Learning models, especially in problems such as image and video tagging, automatic translation, optical character recognition, speech-to-text and others, collectively known as *Computer Perception*. Those improvements have been motivated by the greater computational power developed during those years by several improvements in computing hardware and software and the great amount of data available in the so called *era of Big Data*, but those are not the only reasons for it. The development of *Deep Learning*, term that refers to modern artificial neural networks that employ a series of relatively recent techniques (initialization, activation, regularization, etc.), has been, probably, the most important factor of all. These connectionist Machine Learning models are not only universal approximators, but have a flexible architecture that can be adapted to different types of data and loss functions. Examples of this are convolutional neural networks and recurrent neural networks, adapted to data with spatial structure and data with temporal structure, respectively.

This thesis, structured as a compendium of articles, presents three developments tightly related to Deep Learning that have resulted in the several publications included. In the first place, an application of convolutional neural networks to the problem of prediction in renewable energies is proposed, taking advantage of the spatial structure present in such data, that results in an improvement over the previous results while keeping computational costs under control thanks to the efficiency of artificial neural networks. The related publications were among the first contributions at the time to make use of the new DNN frameworks for renewable energy prediction.

In the second place, the usefulness of Deep Learning models for feature transformation is shown. The importance of a correct feature transformation can be paramount when confronting a Machine Learning problem. Such algorithms can be used as a first step in the modelization pipeline, prior to a classifier or regressor, for example, and split the problem into two more manageable sub-problems: obtaining a good representation of the data and generating the actual prediction from it. In this case, the feature transformation technique for classification problems known as Fisher's Discriminant Analysis (FDA) will be studied. Once the theoretical framework is set up, the limitations and drawbacks of such tool can be more deeply analyzed. Such limitations include its linear nature, which implies a limited expressive power. This is often solved with the use of kernels, with the disadvantage of a much higher computational cost, that relegates this technique to the world of *small to medium data*. To overcome such limitations, a partial equivalence between the traditional technique and Least-Squares based models is exploited. This equivalence will allow to train linear transformers with an iterative algorithm, and, by extension, to use Artificial Neural Networks as the underlying *computational engine*, obtaining non-linear transformations in the process while maintaining reasonable computing costs even for big datasets. Additionally, the use of neural networks in imbalanced classification problems, an application closely related to FDA, will be presented.

In the third place, the application of margin loss functions, such as those usually employed in support vector machines, over artificial neural networks will be studied. As it will be shown, these loss functions can represent a noticeable improvement in the model performance in certain cases, even when highly sophisticated neural networks as LeNet or ResNet are used, and again the efficiency of neural networks in terms of computational cost can be an important advantage over more classical techniques. Also, an adaptation of Deep Learning models to the use of several simultaneous loss functions, margin loss functions among them, which will again produce noticeable changes in the quality of predictions, will be detailed.

Resumen

La década de 2010 a 2020 ha visto una serie de impresionantes mejoras en el rendimiento de los modelos de Aprendizaje Automático, especialmente en problemas como el etiquetado de imágenes y vídeo, la traducción automática, el reconocimiento óptico de caracteres, el reconocimiento del habla y otros, a los que se ha llamado colectivamente *Computer Perception*. Estas mejoras han sido en gran parte motivadas por la mayor capacidad de cálculo desarrollada durante estos años mediante mejoras en el hardware y software de computación y la gran cantidad de datos de los que se dispone en esta llamada *era del Big Data*, pero no han sido estos los únicos motivos. El desarrollo del llamado *Deep Learning*, término que se refiere a las redes neuronales artificiales modernas con las que se emplean una serie de técnicas relativamente recientes (de inicialización, activación, regularización, etc.), ha sido probablemente el factor más importante de todos. Estos modelos conexionistas de Aprendizaje Automático no sólo son aproximadores universales, sino que tienen una arquitectura flexible que se puede adaptar a distintos tipos de datos y funciones de pérdida. Ejemplos de esto son las redes convolucionales y las redes recurrentes, adaptadas a datos con estructura espacial y datos con estructura temporal, respectivamente.

En esta tesis, en formato de compendio de artículos, se presentan tres desarrollos estrechamente relacionados con el Deep Learning y que han resultado en las diversas publicaciones que se incluyen. En primer lugar se presenta la aplicación de redes convolucionales a problemas de predicción en energías renovables, aprovechando la estructura espacial que estos datos tienen, lo que permite mejorar los resultados existentes y mantener los costes computacionales bajo control gracias a la eficiencia de las redes neuronales artificiales. Las publicaciones relacionadas estuvieron entre las primeras en hacer uso de las nuevas herramientas de desarrollo de redes neuronales profundas en el campo de la predicción en energías renovables.

En segundo lugar se muestra la utilidad de los modelos de Deep Learning a la hora de realizar transformaciones de variables. Una transformación de variables adecuada puede tener una gran importancia al abordar un problema de Aprendizaje Automático. Este tipo de técnicas se pueden utilizar como un primer paso dentro del proceso de modelización, previo al clasificador o regresor, por ejemplo, y dividen el problema en dos subproblemas más manejables: obtener una buena representación de los datos y generar la predicción a partir de ella. En este caso se estudiará la transformación de variables para problemas de clasificación conocida como Análisis Discriminante de Fisher (FDA). Una vez el marco teórico está dispuesto, las limitaciones y desventajas presentes en esta técnica pueden ser analizadas en profundidad. Estas limitaciones incluyen su naturaleza lineal, que implica un poder expresivo limitado. Esto se resuelve habitualmente con el uso de kernels, con el consiguiente aumento en el coste computacional, lo cual relega a esta técnica al mundo de los *datos de tamaño pequeño y medio*. Para solventar estas limitaciones, se explota una equivalencia parcial entre la técnica tradicional de transformación y los modelos de mínimos cuadrados. Esta equivalencia permite entrenar transformadores lineales con un algoritmo iterativo y, por extensión, el uso de Redes Neuronales Artificiales como *motor de cálculo* subyacente, obteniendo transformaciones no lineales y a la

vez manteniendo los costes computacionales en niveles razonables incluso para grandes conjuntos de datos. Adicionalmente se muestra el empleo de redes neuronales a problemas de clasificación desequilibrada, aplicación estrechamente relacionada con FDA.

En tercer lugar se estudia la aplicación de funciones de pérdida con margen, iguales a las empleadas habitualmente con máquinas de vectores de soporte, en redes neuronales artificiales. Como se podrá comprobar, estas funciones de pérdida pueden suponer una sensible mejora en el rendimiento del modelo en ciertos casos, incluso cuando se emplean redes neuronales altamente sofisticadas como las LeNet o ResNet, y de nuevo la eficiencia en términos de coste computacional de las redes neuronales puede suponer un importante punto a favor de estas técnicas respecto de otras más clásicas. También se describe la adaptación de modelos de Deep Learning al uso de varias funciones de pérdida simultáneas, entre ellas las funciones con margen, lo que en determinados casos también afectará notablemente a la calidad de los resultados.

Acknowledgements

In the first place, thanks to all the readers of this thesis, and specially to the members of the committee, the colleagues that have collaborated in the related publications, and those who have helped in fixing and improving this thesis with their valuable opinions.

I would like to thank Professor José R. Dorronsoro for sharing with me his expert knowledge and wisdom, and for guiding me through this too long and often steep path.

I also want to thank all my co-workers and friends at Instituto de Ingeniería del Conocimiento. I've learnt a lot from them, and embarking on this adventure would have never been possible without their understanding.

My friends and relatives also deserve recognition, since I have, more often than I would have liked to, given them less attention than they deserved, and nevertheless, there they are, still by my side.

And finally, my deepest gratitude to my parents, who have given me all that I have, my brother, Jesús, and his family (Eva and the little Daniel), and Sandra, that keeps supporting me even though I don't even get along with myself sometimes.

To my father, whom cancer has taken away too soon during the writing of this thesis.

Agradecimientos

En primer lugar, gracias a todos los lectores de esta tesis, y en especial a los miembros del tribunal, los compañeros que han colaborado en las publicaciones relacionadas y todos aquellos que han ayudado a corregir y mejorar esta tesis con sus valiosas opiniones.

Me gustaría agradecer al profesor José R. Dorronsoro el que haya compartido conmigo su experto conocimiento y su sabiduría, y el haberme guiado a través de este camino demasiado largo y a menudo difícil.

También quería dar las gracias a mis compañeros de trabajo y amigos en el Instituto de Ingeniería del Conocimiento. He aprendido mucho de ellos, y embarcarme en esta aventura nunca habría sido posible sin su comprensión.

Mis amigos y familiares también merecen reconocimiento, ya que, más a menudo de lo que me habría gustado, les he dedicado menos atención de la que merecen, y, aun así, ahí siguen a mi lado.

Y finalmente, mi más profundo agradecimiento a mis padres, que me han dado todo lo que tengo, a mi hermano, Jesús, y su familia (Eva y el pequeño Daniel), y a Sandra, que me sigue soportando aunque a veces ni yo mismo me aguanto.

A mi padre, a quien el cáncer se ha llevado antes de tiempo durante la redacción de esta tesis.

Symbols

N	number of observations in the dataset
n	observations index
d	dimension of the input space
X	$N \times d$ input matrix
d'	dimension of the target space
Y	$N \times d'$ target matrix
w	$d \times 1$ parameter vector
c	dimension of the latent feature space
W	$d \times c$ parameter matrix
$\nabla_w f$	gradient of function f with respect to vector w
$\nabla_W f$	gradient of function f with respect to matrix W
$tr(A)$	trace of matrix A
A^t	transpose of matrix A
$SVD(A)$	Singular Value Decomposition of matrix A
$k(x, x')$	kernel function applied to patterns x and x'
D	dimension of the kernel space
$\mathcal{O}(\cdot)$	Landau Big O

Chapter 1

Introduction

1.1 Machine Learning Overview

Although basic statistics, mostly descriptive, have been used since the beginning of Civilization (Thucydides' *History of the Peloponnesian War*), and the earliest known use of Statistical Inference dates from the 9th century (Al-Kindi's *Manuscript on Deciphering Cryptographic Messages*), modern Statistics only has a few centuries, and its development is closely related to the rise of European sovereign states and the growing interest in census and other demographic studies.

Machine Learning, also called Statistical Learning, could be considered a subfield of Statistical Inference, since its methods analyze data to deduce properties of the underlying probability distribution in order to be able to make prediction over unseen data drawn from the same population.

But Machine Learning (Figure 1.1.1) is also considered a form of Artificial Intelligence, and as such, it sought the development of systems capable of perceiving the environment and taking actions accordingly to maximize the chance to achieve a goal. Artificial Intelligence encompasses other subfields, like Symbolic Artificial Intelligence, that compete with and complement each other and have enjoyed different levels of success in the last decades.

For instance, it is generally accepted that the dominant branch of Artificial Intelligence during the 70s and 80s of the 20th century was Symbolic Artificial Intelligence, with the development of Expert Systems that emulate the human ability of decision making by using knowledge databases usually represented as collections of *if-then* rules. This approach to Artificial Intelligence produced successful Expert Systems like *MYCIN* [1], designed to diagnose infectious diseases and recommend personalized treatments based on the patient's medical data. Expert Systems have, however, a serious limitation: the process of acquiring knowledge is difficult and costly, and requires the very intensive involvement of human experts.

In this context, Statistical Learning arises as a more practical alternative by only requiring data, but no knowledge databases, to operate, with the only significant drawback being perhaps the great computing power needed. Thus, during the 90s of the 20th century, the focus on Artificial Intelligence progressively shifted from knowledge-based techniques to data-based techniques. So, the Statistical Learning techniques are focused on making the machine *learn* to solve problems, understanding

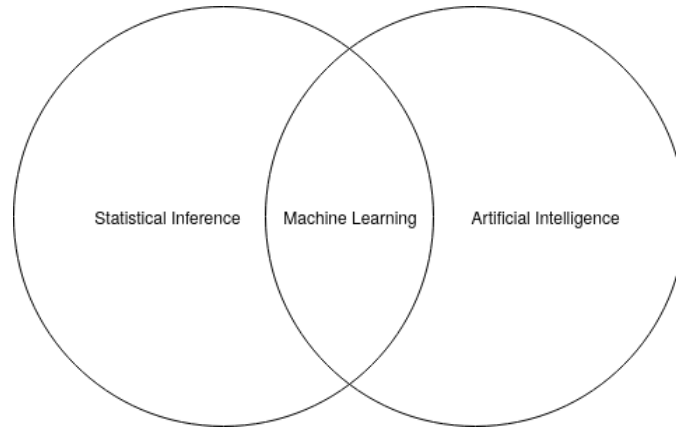


Figure 1.1.1: Machine Learning at the intersection of Statistical Inference and Artificial Intelligence

learning as an improvement of the performance with the use of samples of data.

A *learning machine* is defined by the relation

$$t : D \mapsto m,$$

where D is the training dataset, m is a predictive model, and t is a training algorithm, that uses D to adjust m according to some fitness criteria. These three entities, data, model, and training algorithm, will have a prominent role in what follows.

1.1.1 Data

A dataset D of N samples consists in a matrix $X \in \mathbb{R}^{N \times d}$, the set of d -dimensional observed features, and, often but not always, a matrix $Y \in \mathbb{R}^{N \times d'}$, the set of d' -dimensional targets. Each row of X corresponds to a sample and each column corresponds to a feature, while each row of Y corresponds to a sample and each column to a dimension of the target. Often, d' will be exactly 1, and in such cases Y might be also denoted by y .

Although most predictive models and optimization algorithms are designed to operate exclusively with purely numerical X , real world data is often categorical. Fortunately, usually a straightforward encoding of the categorical data, like the well known one-hot encoding, is often enough to overcome this issue. In some rare cases, though, resorting to a more elaborated embedding of these features is mandatory to preserve all the valuable information they contain.

On the other hand Y can adopt many forms, or even be non-existent, and Machine Learning techniques are usually grouped [2, 3] according to the following taxonomy based on the properties of the data, and more precisely on the availability and form of Y :

- Supervised Learning: when the target Y is known. That is, the training dataset is labelled. Contains several subcategories, like
 - Regression: when Y has numerical values
 - Classification: when Y is a categorical variable

Table 1.1.1: Iris flower dataset

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...

- Ordinal regression/classification: somewhat in between regression and classification, when Y is an ordinal variable. That is, discrete but ordered.
- Supervised feature selection/transformation: when Y is used to filter or transform the original features.
- Unsupervised Learning: when the target Y is not known. That is, the training dataset is not labelled. Also several subcategories are included here, for example
 - Clustering: when the objective is to group the data according to a given similarity metric.
 - Anomaly detection: when the objective is to detect outliers.
 - Unsupervised feature selection/transformation: when filtering or transforming the original features is the goal, and no target is used for the purpose.
- Semi-supervised Learning: a mix of supervised and unsupervised learning [4]. Usually, it arises when only a limited amount of data is labelled, but a large amount of unlabelled data is available. It is common to use the unlabelled data to pretrain the predictive model, and the labelled data to fine-tune it.
- Reinforcement Learning: when no labelled training dataset is available, but the environment provides some kind of positive/negative feedback on the predictions, and a balance between the exploration of the unknown space and the exploitation of the known space is sought [5].
- Motivational Learning: when a positive/negative feedback is present but very scarce, and the learning system has to use some internally defined drive functions to motivate curiosity and learn autonomously [3].

Most of what follows will be focused on supervised learning, specially on techniques applied for classification, regression and feature transformation. As an example, lets take into account the celebrated *Iris flower* dataset, introduced by Ronald Fisher in [6]. The dataset has 150 samples with 4 independent variables each, sepal length, sepal width, petal length and petal width (Table 1.1.1), and the target is available and takes categorical values (setosa -0-, vericolor -1- or virginica -2-, Figure 1.1.2); so, this is an example of the supervised learning setup, and, more precisely, classification.

Another well known example is the Boston Housing dataset. This dataset contains U.S Census Service information on housing in Boston, Massachusetts. The

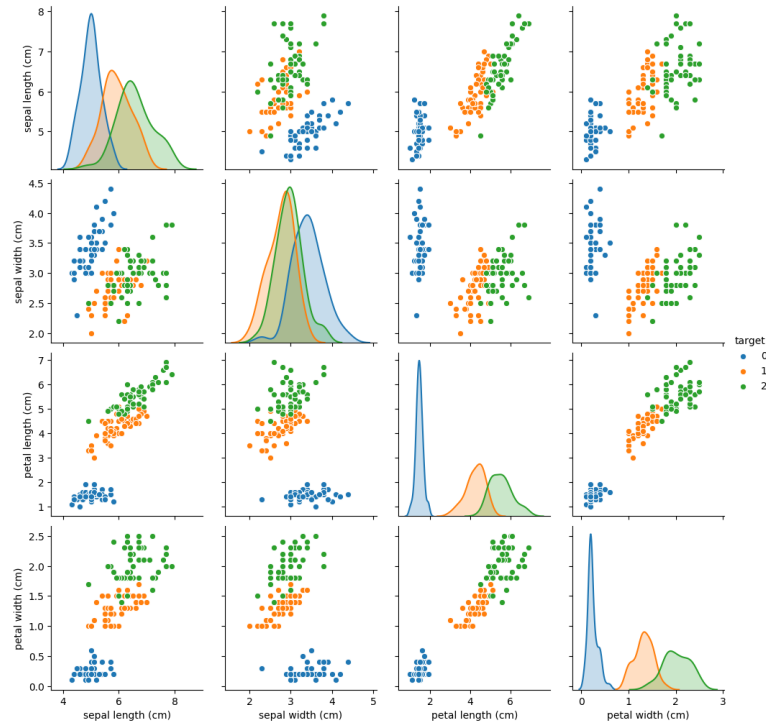


Figure 1.1.2: The Iris flower dataset, with 4 features and 3 classes represented with colors

target is the median value in thousands of the homes in each Boston area, and the following 13 independent variables (Table 1.1.2) are provided for the task:

- Per capita crime rate by area (CRIM)
- Proportion of residential land zoned for lots over 25,000 sq.ft. (ZN)
- Proportion of non-retail business acres per area (INDUS)
- Charles River indicator (= 1 if tract bounds river; 0 otherwise) (CHAS)
- Nitrogen oxide concentration (parts per 10 million) (NOX)
- Average number of rooms per dwelling (RM)
- Proportion of owner-occupied units built prior to 1940 (AGE)
- Weighted mean of distances to five Boston employment centres (DIS)
- Index of accessibility to radial highways (RAD)
- Full-value property-tax rate per 10,000\$ (TAX)
- Pupil-teacher ratio by area (PTRATIO)
- $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by area (B)
- Lower status of the population (percent) (LSTAT)

Table 1.1.2: Boston Housing dataset

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...

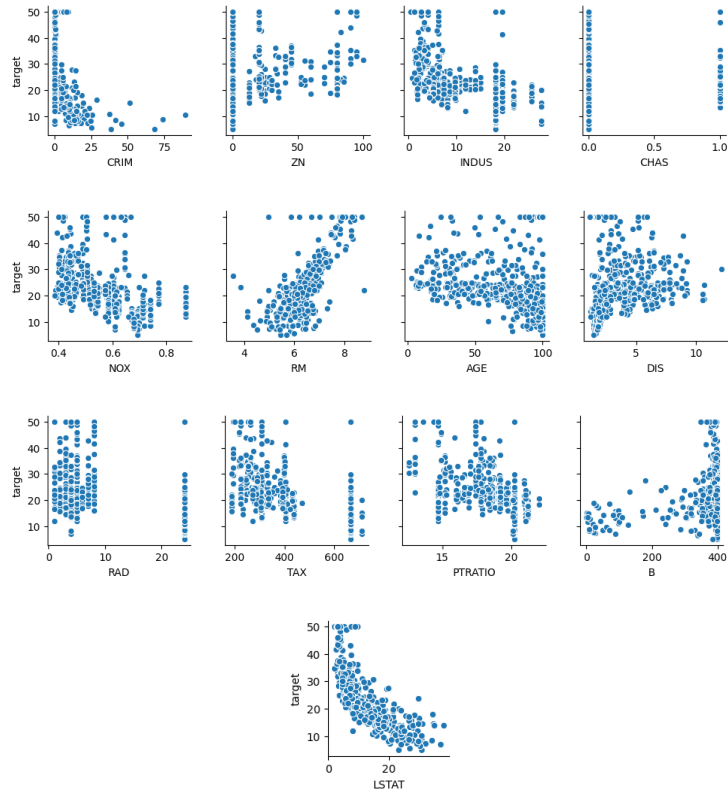


Figure 1.1.3: The Boston housing dataset, its 13 features plotted against the target

It has 506 samples and, since the target is numeric (Figure 1.1.3), it is an example of a regression dataset.

In what follows, let $\{(x_n, y_n)\}$, $n = 1 \dots N$, be a set of observation–target pairs, and $X \in \mathbb{R}^{N \times d}$ and $Y \in \mathbb{R}^{N \times d'}$ the matrices where each row represents an observation or target, respectively. As mentioned, $d' = 1$ usually, but that will not always be the case and it is not uncommon to find regression problems with $d' > 1$. In classification problems, Y is a 1–dimensional column vector of labels, and might need to be reencoded (as in one–hot encoding) before been fed to a classifier. Also, in an unsupervised learning setting, Y will not be available.

In all the cases stated above, it is assumed that both X and, if in a regression setting, Y have been transformed in a previous step by removing their mean, that is, by removing

$$\bar{X} = \frac{1}{N} \sum_{n=1}^N x_n, \quad \bar{Y} = \frac{1}{N} \sum_{n=1}^N y_n,$$

from every row–vector of X and Y , respectively.

1.1.2 Model

A predictive model will map input features to predictions:

$$m : \mathbb{R}^d \rightarrow \mathbb{R}^{d'} \quad (1.1.1)$$

where \mathbb{R}^d is the d -dimensional feature space and $\mathbb{R}^{d'}$ the d' -dimensional target space.

Since every predictive model implies some assumptions on the data, many families of them exist, and there is no single one that works best on all problems [7]. Regarding the internal structure derived from the assumptions made, and in particular the properties of their parameter set, predictive models can be classified in the following way:

- Parametric models: those that have an internal parameter set $\theta \in \Theta$, with Θ the space of feasible parameter sets, whose structure and size is (almost) independent of the data. This category includes multilayer Artificial Neural Networks with a fixed architecture, for example, whose internal parametric structure, first hidden layer and output layer aside, is independent of the data. Usually, these models have a fixed computational prediction cost, regardless of the size of the dataset used to train them, which is a great advantage in the era of Big Data.
- Non-parametric models: those whose structure is determined by the data. Instance-based models like Nearest Neighbors are non-parametric models.
- Semi-parametric models: which combine characteristics of the two categories above. Decision Trees and Kernel Methods such as Support Vector Machines are in this category [8].

Other taxonomies that share some of these categories have been proposed [9].

The *k-Nearest Neighbors* (k-NN) is perhaps the most intuitive of all Machine Learning models, and arguably the canonical example of non-parametric model. It assumes a data sample target will be similar to those of samples located in its neighborhood in the feature space. k-NN can be adapted to both classification (k-NN classifier) and regression (k-NN regressor), and is closely related to *k-Means* which is used for clustering and makes similar assumptions.

The simplest k-NN model has only two hyper-parameters: the metric used to measure distances between samples, usually the Euclidean distance, and k , the number of neighbors to consider for inference.

As any other Machine Learning model, it has to be fitted to the data before being used for prediction, but in this particular case, the training process is essentially as simple as storing the whole training dataset, including both features and targets. Once fitted, predicting the target of a new, unseen pattern consists on:

- A majority voting among the k nearest neighbors selected from the stored training data for classification.
- The mean of the target for the k nearest neighbors selected from the stored training data for regression.

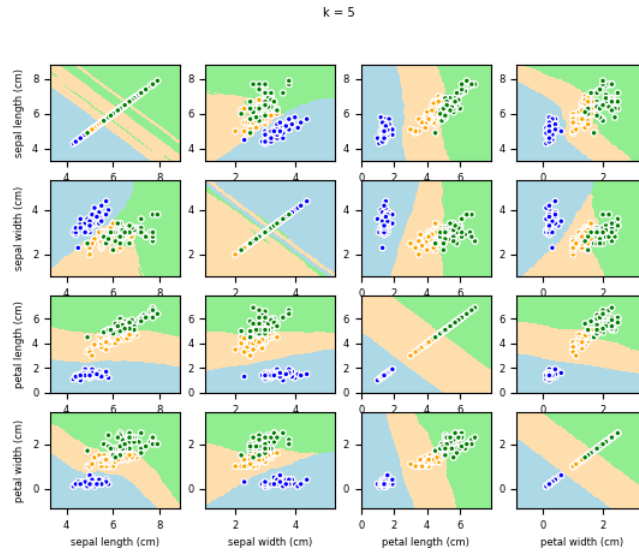


Figure 1.1.4: k-NN (using only pairs of features) boundaries for Iris flower dataset with $k=5$

Thus, if, for example, the chosen metric is Euclidean and $k = 5$, the kNN model applied to the Iris dataset will classify unseen data according to the boundaries in the space of independent variables represented in Figure 1.1.4. When applied to each feature of the Boston Housing regression dataset independently, the kNN model with $k = 5$ produces the results displayed in Figure 1.1.5.

Alternatively, a parametric model could be used, and the simplest of them are *Linear Models*, like *Linear Regression* (for regression) and *Logistic Regression* (for classification), although they make more assumptions on the data than the neighbor models.

If a logistic regression model is trained on the Iris dataset, the classification boundaries produced will be linear, as can be seen in Figure 1.1.6. Also, the linear regression model can of course be applied to the Boston dataset producing the regression lines of Figure 1.1.7.

Another important element to consider is the use of feature selection or transformation as a first component in the model *pipeline*. Focusing on transformation, widely known examples are:

- *Principal Component Analysis* (PCA)
- *Fisher Discriminant Analysis* (FDA)
- *Partial Least Squares* (PLS)
- *Canonical Correlation Analysis* (CCA)

The effectiveness of such methods has been proved with models like *Principal Component Regression* (PCR) [10], that combines the PCA transformation with a Linear Regression and has properties similar to those of Ridge Regression.

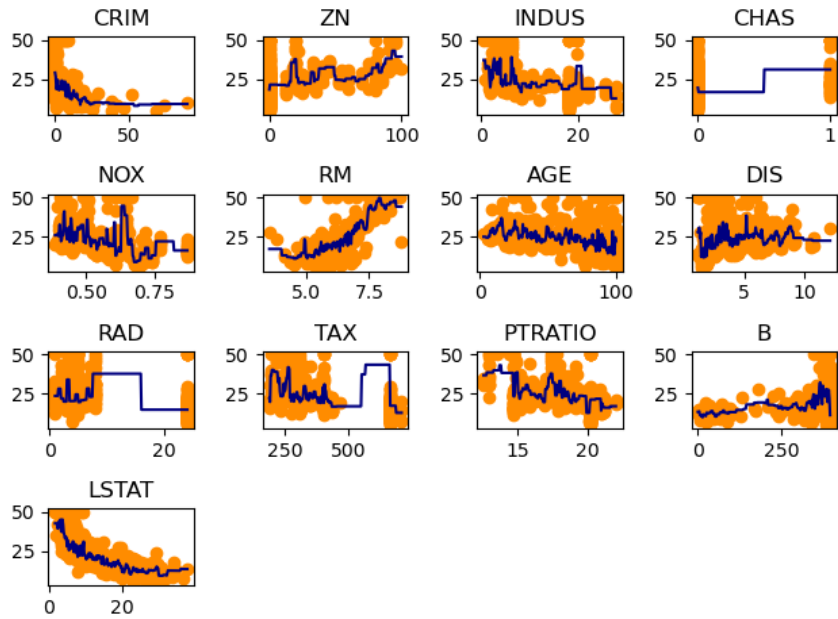


Figure 1.1.5: k-NN (using single features) predictions for Boston housing dataset with $k=5$. Orange dots represent actual data and the dark blue line represents the predictions.

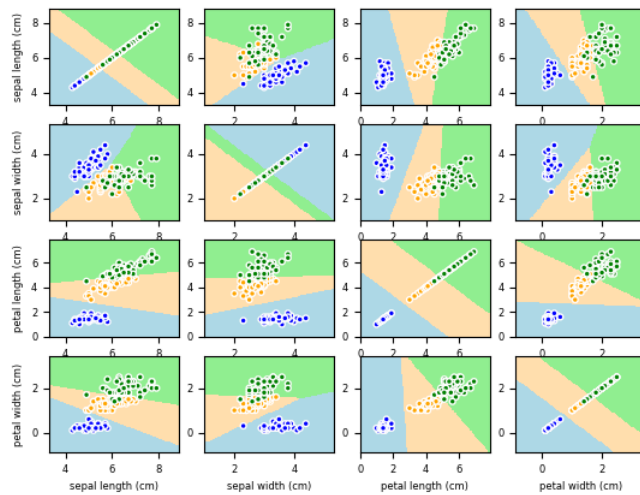


Figure 1.1.6: Logistic Regression (using only pairs of features) boundaries for Iris flower dataset

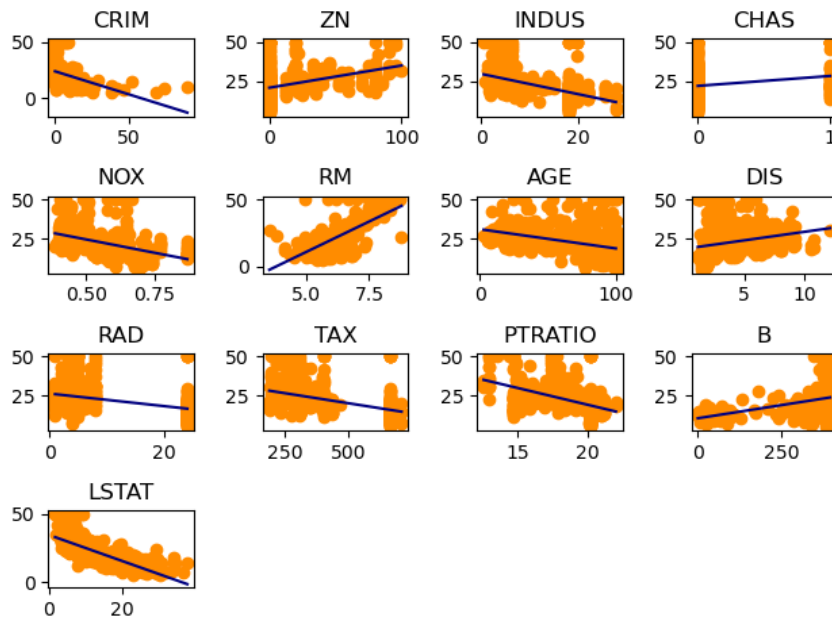


Figure 1.1.7: Linear Regression (using single features) predictions for Boston housing dataset. Orange dots represent actual data and the straight, dark blue line represents the predictions.

However, many transformation techniques are linear, which implies a limited expressive power, and, more importantly, have a direct training algorithm that produces an exact solution, which results in bad scalability with the size of the dataset as will be detailed later. The use of kernels to mitigate the first shortcoming is common, but solutions to the second are not widespread as of today, which limit the use of such tools. On the other hand, feature transformation has some interesting qualities, like the possibility to decompose a bigger problem into simpler subproblems in a simple and elegant way.

1.1.3 Training

Usually, the prediction $m(x) = \hat{y}$ is expected to approximate the target $y \in \mathbb{R}^d$, which is achieved by making the training algorithm t optimize an objective function.

Often, this consists on minimizing a loss function $L(Y, \hat{Y})$ between the predictions \hat{Y} and the targets Y , possibly taking some restrictions into account, but the alternative approach of maximizing a fitness function is also quite common.

Thus, a training algorithm, or optimizer, seeks to find the set of parameters θ of m in the space Θ of valid parameters minimizing a function L :

$$\begin{aligned} \min_{\theta} \quad & L(Y, \hat{Y}) \\ \text{s.t.} \quad & \theta \in \Theta. \end{aligned}$$

The computational complexity of such an optimization problem is highly dependent on the mathematical properties of the loss function. Smooth and convex func-

tions are, naturally, easier to optimize than non-smooth and non-convex functions, since the existence of local optima and plateaus often causes difficulties.

In some special occasions a direct method is available and the solution of the optimization problem is deterministic and can be calculated analytically. This is the case, for example, of Ordinary Least Squares, a method that fits a Linear Regression model in order to minimize the residual sum of squares (RSS):

$$\begin{aligned} L(Y, \hat{Y}) &= \sum_{n=1}^N (y_n - \hat{y}_n)^2 \\ &= \|Y - \hat{Y}\|^2. \end{aligned}$$

Since $\hat{Y} = m_\theta(X)$ is linear in terms of θ , L is quadratic in terms of θ , and hence smooth and convex with a single global minimum. So, the minimum of L is the only point that meets the condition

$$\nabla_\theta L = 0,$$

and therefore solving this equation leads to the optimal θ .

This, however, is not the case in general, and most optimization problems don't have a known analytic solution. Also, direct methods present a serious drawback: learning is not incremental, i.e., if new data is incorporated into the training dataset, the whole training process must be repeated from the ground up.

On the other hand, iterative methods, like Stochastic Gradient Descent (SGD), yield incremental solutions that can be calculated numerically and are usually less costly for big size datasets.

To be suitable for an incremental learning setup, a loss function $l(y, \hat{y})$ has to be applicable to single instances of the dataset. Some of the most common of such functions for regression are outlined in table 1.1.3, while those more commonly used for binary and multiclass ([11]) classification are listed in tables 1.1.4 and 1.1.5, respectively. In the case of cross-entropy loss the targets and predictions of the C -class classification problem are one-hot encoded. Observe that some of the outlined loss functions are not derivable at all points.

Table 1.1.3: Regression losses

Quadratic	$(y - \hat{y})^2$
Absolute	$ y - \hat{y} $
ϵ -insensitive	$\max(0, y - \hat{y} - \epsilon)$

Table 1.1.4: Binary classification losses

Cross-entropy (logistic)	$-y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$
Hinge	$\max(0, 1 - y\hat{y})$

In the case of a linear model minimizing a quadratic loss, an iterative method will have, if certain conditions on the data are met and the step size (learning rate)

Table 1.1.5: Multiclass classification losses

Cross-entropy	$-\sum_{c=1}^C y_c \log(\hat{y}_c)$
Multiclass hinge	$\max(0, 1 + \max_c((1 - y_c)\hat{y}_c) - \max_c(y_c\hat{y}_c))$

of the algorithm is adequate, little trouble finding the optimal set of parameters: ensuring that each step of the iterative algorithm goes in the opposite direction of the gradient $\nabla_W l$, guarantees that the optimal point, that is the only minimum of the function, will be reached eventually.

SGD-based iterative methods can also optimize more complex functions, even when non-linear predictive models are used, but in such cases some difficulties will arise due to the non-convex or non-smooth nature of the function itself. Optimization over such non-convex functions requires a more elaborated strategy to avoid getting trapped in local minima or plateaus, but even so, these methods have become the gold standard when dealing with highly complex non-linear models.

To wrap up, regarding the kind of optimization technique used to find the optimal configuration of a selected predictive model for a given dataset, training algorithms can be placed into one of two categories:

- Direct: closed-form analytic methods that yield exact solutions, usually with the serious drawback of a bad scalability with the size of the data.
- Iterative: incremental, numerical methods that yield approximate solutions, usually without guaranteed convergence to the optimum but more scalable than direct methods for big datasets.

1.2 Methodology

1.2.1 The Components of a Loss

Reformulating the ideas of [12, 7, 13], a supervised learning problem is defined by the space of pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$, endowed with a probability distribution $P(x, y)$, where the sought relationship between \mathcal{X} and \mathcal{Y} , and the conditional probability $P(y|x)$, is unknown.

The objective is to find the model m that minimizes a cost function, the *expected error* or *expected risk*,

$$E(m) = \int l(y, \hat{y}) dP(x, y),$$

where $\hat{y} = m(x)$ and the loss l measures discrepancy between the model m predictions and the targets y . The model sought is

$$m^* = \operatorname{argmin}_m E(m).$$

The quality of the model may, however, be considerably affected by many factors present outside the ideal mathematical world. The available dataset might have missing values, measurement errors, or simply be too small to faithfully represent

$\mathcal{X} \times \mathcal{Y}$. The choice of model hyper-parameters may be poor, or even the model assumptions might be incorrect. The hyper-parameters related to the training algorithm could also lead it to a sub-optimal solution, the algorithm itself might not be a wise choice for a particular objective function, or a too tight time constraint could force the optimizer to stop before convergence.

1.2.1.1 Estimation

In practice, only a finite sample (X, Y) of N data, defined by the pairs $(x_n, y_n), n = 1 \dots N$, is available. The *empirical error* or *empirical risk* to be minimized is then

$$E_N(m) = \frac{1}{N} \sum_{n=1}^N l(y_n, \hat{y}_n),$$

and the best model according to this error is

$$m_N^* = \operatorname{argmin}_m E_N(m).$$

This limitation on the available data not only affects the quality of the empirical error as an approximation of the expected error, but also leads to what is known as *the curse of dimensionality* [13, 14] as the number of available feature increases.

The *generalization error*, also called *test error*, of a predictive model is the prediction error over independent test data, while the prediction error over the data used to train the model is known as *training error* [13]. The goal of the quality assessment process is to correctly estimate the generalization error of the trained model. Unfortunately, the training error is not a good estimate of the generalization error and so, it is a methodological mistake to assess a model on the same data that was used for training it.

Some datasets specify train and test partitions, in which case it is advisable to train the predictive model using the train partition and assess its quality with the test partition, not only to avoid being deceived by the usually optimistic metrics calculated over the training partition, but also to be able to compare results with other models on the same dataset. However, most datasets do not specify any kind of partition. In those cases, probably the most cautious way to proceed is to perform Cross-Validation (CV). The most common form of non-exhaustive CV, *k*-fold CV, starts by randomly partitioning the dataset into *k* equal sized subsamples. Once the dataset is partitioned, a single subsample of the *k* is saved as the validation data, while the remaining *k* - 1 subsamples are used as training data. This process is repeated *k* times, with each of the *k* subsamples used exactly once as the validation data, and the *k* results are then averaged to produce a single estimation of the generalization error.

1.2.1.2 Approximation

Not only the data is limited in the real world, but also the time available to train different predictive models. So, only a family \mathcal{M} of models can be explored, and since the optimal m^* is unlikely to be part of \mathcal{M} , the best model available will be denoted

$$m_{N, \mathcal{M}}^* = \operatorname{argmin}_{m \in \mathcal{M}} E_N(m).$$

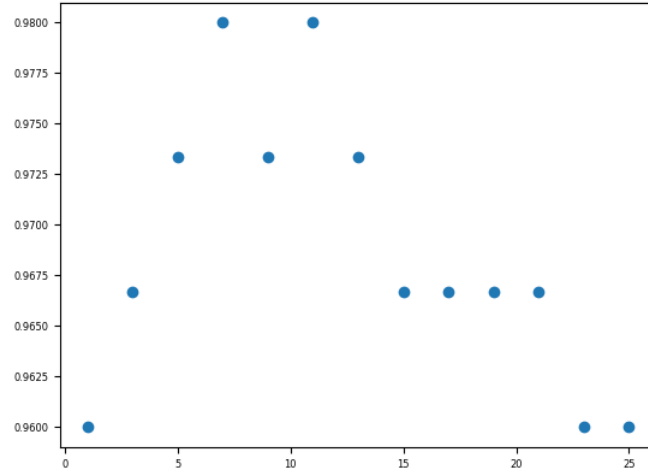


Figure 1.2.1: k-NN cross-validation scores for different values of k for the Iris flower dataset

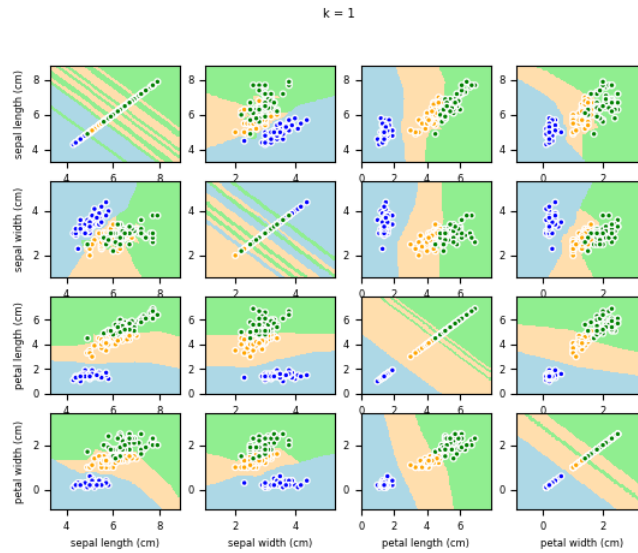


Figure 1.2.2: k-NN (using only pairs of features) boundaries for the Iris flower dataset with k=1

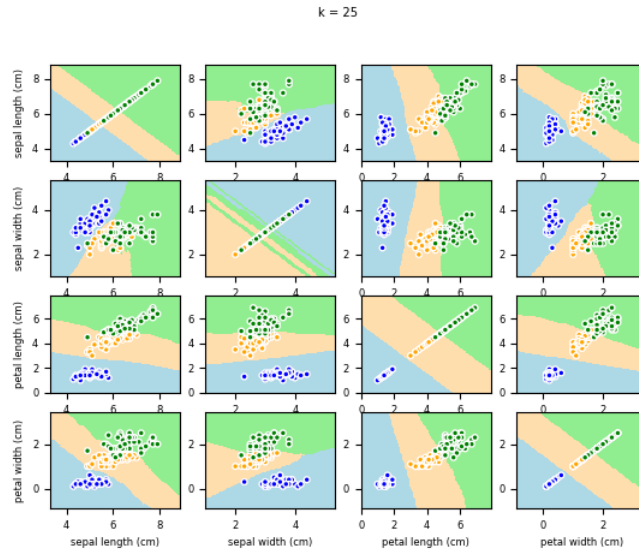


Figure 1.2.3: k-NN (using only pairs of features) boundaries for the Iris flower dataset with $k=25$

Moreover, even if just a single model is to be considered, the space defined by its hyper-parameters is usually too big to be exhaustively explored, with the subsequent loss of performance induced by a bad choice of hyper-parameter values. As an example, figure 1.2.1 shows different scores (calculated by cross-validation) of the k-Nearest Neighbor classifier as a function of k for the Iris dataset. It seems the best values for this hyper-parameter lie between 5 and 13, forming a sort of 'inverted U' curve. This is a visual example of what is known as the *bias-variance tradeoff* [13, 15]: if the model is too inflexible (high k , figure 1.2.3), the bias component of the error will be too high, while if the model is too flexible (low k , figure 1.2.2), the variance component of the error will rise.

Hyper-parameter search adds another layer of complexity to the cross-validation methodology explained before, since manually choosing the hyper-parameters that minimize the error biases the model and yields an overly-optimistic score. The recommended approach in this case is known as *nested cross-validation* [16]. At its higher level, the experiment consists again in fitting the model to the data and performing an assesment; but at its lower level, training is a more involved process where several hyper-parameter sets are tried and evaluated, and only the best performing one is then used for the upper level assesment. Each hyper-parameter set is evaluated in an inner CV loop, while an outer loop (with only one iteration if a train-test partition is available or k iterations of k -fold CV if it is not) determines the generalization error of the model and its hyper-parameter search.

Several strategies can be used for the hyper-parameter optimization: from the brute force grid search and randomized search [17] to the more sophisticated bayesian search [18], this last one being of special interest when the underlying model whose hyper-parameters must be optimized is computationally complex and the waste of time exploring bad hyper-parameter sets is unacceptable.

1.2.1.3 Optimization

Finally, a direct algorithm that provides an analytical solution for the training of a predictive model is not generally available or is available but impractical due to its cost for a given training dataset, and iterative algorithms can only provide numerical solutions that approximate the analytical one. Since only a limited set of training algorithms and their hyper-parameters, \mathcal{T} , can be explored, m^* will not be found in general. Then, the best available model considering only training algorithms in \mathcal{T} will be denoted

$$m_{N,\mathcal{M},\mathcal{T}}^* = \operatorname{argmin}_{m \in \mathcal{M}, t \in \mathcal{T}} E_N(m_t).$$

with subindex t of m_t denoting the effect of the training algorithm on the final performance of the model.

Usually, if a model is expressive enough, that is, it has enough complexity to *learn* the data, the training error decreases with each iteration of the training algorithm. But this often comes with an increase of the test error if too many iterations are performed. This is known as *overfit*, and implies that the model is learning not just the relevant information in the data but also the noise, therefore losing generalization capability. A good way to avoid this is to partition the training data into train and validation sets, using the first to fit the model and the second to monitor the error function at each iteration of the training algorithm. Once the error over the validation data ceases to decrease for more than just a few iterations, it is advisable to finish the training since the model is not improving its generalization capability. This technique is known as *early stopping*, and is widely used in SGD-based training algorithms [19].

1.2.1.4 Combining the three components

At this stage, it should be clear that selecting an adequate model and fitting it successfully to the data is not trivial, given the number of factors to consider.

Also, datasets are usually finite, and so is the time available, which implies that the number of models to be considered and the number of iterations to fit each one of them are also limited. Then, the three error components cannot be minimized at the same time and a compromise between them must be sought.

Summing up what was discussed in the previous section, the excess error of a fitted model can be decomposed as

$$\begin{aligned} \mathcal{E} &= \mathbf{E}(|E(m^*) - E(m_N^*)|) + \\ &\quad \mathbf{E}(|E(m_N^*) - E(m_{N,\mathcal{M}}^*)|) + \\ &\quad \mathbf{E}(|E(m_{N,\mathcal{M}}^*) - E(m_{N,\mathcal{M},\mathcal{T}}^*)|) \\ &= \mathcal{E}_{est} + \mathcal{E}_{app} + \mathcal{E}_{opt}, \end{aligned}$$

where the *estimation error*, \mathcal{E}_{est} , measures the impact of minimizing the empirical error instead of the expected error, the *approximation error*, \mathcal{E}_{app} , measures *how far* the optimal model is from model family \mathcal{M} , and the *optimization error*, \mathcal{E}_{opt} , measures the influence of the optimization algorithm on the quality of the solution.

1.2.2 Metrics

Moreover, demonstrating that a model has value in the real world generally requires a metric that is interpretable in business terms, something that the usual loss functions do not accomplish. Notice that such metrics are only applied a posteriori once the model training is finished. Therefore they are independent of the training algorithm and don't need to be applicable to single instances of the dataset, just to the whole test partition. Although such metrics are not used during the training of the model, they can, however, be used to guide a hyper-parameter search, since they can be applied to a cross-validation fold just in the same way they would be applied to a test partition.

The choice of a metric might be determined by the properties of the training dataset or the behavior desired for the model. For example, a particular metric can be better suited for a dataset that includes a considerable number of outliers, while other metric might be a better choice for a more uniform dataset, even when the targets have the same form in both cases. Conversely, a metric might be the best choice in a classification setup where it is critical not to miss false negatives, while a different metric could be the best choice if, using the same dataset, it is more desirable to avoid false positives.

Tables 1.2.1 and 1.2.3 display some of the most commonly used metrics for regression and classification problems, respectively. The notation used in 1.2.3 is explained in table 1.2.2.

Table 1.2.1: Regression metrics

Mean Squared Error	$MSE(Y, \hat{Y}) = \frac{1}{N} \sum_{n=1}^N (y_n - \hat{y}_n)^2$
Mean Absolute Error	$MAE(Y, \hat{Y}) = \frac{1}{N} \sum_{n=1}^N y_n - \hat{y}_n $
R^2 Score/Coefficient of Determination	$R^2(Y, \hat{Y}) = 1 - \frac{\sum_{n=1}^N (y_n - \hat{y}_n)^2}{\sum_{n=1}^N (y_n - \bar{y})^2}$ where $\bar{y} = \frac{1}{N} \sum_{n=1}^N y_n$
Explained Variance	$EV(Y, \hat{Y}) = 1 - \frac{Var(Y - \hat{Y})}{Var(Y)}$

Table 1.2.2: Classification notation

	Condition positive	Condition negative
Predicted positive	True positive (tp)	False positive (fp) or type I error
Predicted negative	False negative (fn) or type II error	True negative (tn)

Focusing on regression metrics, the most popular are the Mean Squared Error (MSE) and the Mean Absolute Error (MAE). Both of them take values from 0 (perfect fit) to ∞ . MSE is usually favored for its smoothness. It is very sensible to outliers, which can be a problem, and not very discriminative if the residuals are too small. MAE is more stable and resilient to outliers, but is not differentiable in 0, which may result in more complex training algorithms. Another advantage of MAE

Table 1.2.3: Classification metrics

Accuracy	$ACC(Y, \hat{Y}) = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(\hat{y}_n = y_n) = \frac{tp+tn}{N}$ where \mathbb{I} is the indicator function
Precision (Positive Predictive Value)	$PPV(Y, \hat{Y}) = \frac{tp}{tp+fp}$
Recall (True Positive Rate)	$TPR(Y, \hat{Y}) = \frac{tp}{tp+fn}$
$F1$	$F1(Y, \hat{Y}) = 2 \frac{PPV \cdot TPR}{PPV + TPR}$
Area Under the ROC Curve	$AUC(Y, \hat{Y}) = \int_{x=0}^1 \frac{TPR(x)}{FPR(x)} dx$ where $FPR(Y, \hat{Y}) = \frac{fp}{fp+tn}$ is the False Positive Rate

is its interpretability, since the units of MAE are the same as those of targets and predictions.

Less used metrics include Explained Variance and the R^2 , also known as Coefficient of Determination. Explained Variance is rarely used for regressors, but it is common as a transformer metric, for example in PCA or PLS. R^2 is a more elaborated score metric that works like a scale-free version of MSE. It takes values from 1 (perfect fit) to $-\infty$, with 0 being the R^2 of a constant model that predicts the mean target. It can be interpreted, then, as how much better the model is compared to the constant mean model, regardless of the magnitude of the target or its distribution. This is the preferred metric for evaluation of trained regression models in libraries as successful and solid as Scikit-learn [20].

In classification, the choice of a correct metric can be even more influential than in regression. The most popular metric for classification is Accuracy, defined as the fraction of correct predictions, probably because of its simple interpretation, implementation, and the fact that it is applicable for both binary and multiclass targets. It is the default metric for a posteriori evaluation of classifiers in Scikit-learn [20]. However, it is a very poor choice in some very common real world problems, as, for example, imbalanced classification, since it tends to favor classifiers that ignore the least represented classes. Other metrics like Precision, defined as the fraction of true positives in the positives predicted by the model in a binary classification task, and Recall, defined as the fraction of positives tagged as such by the model in a binary classification task, can overcome this limitation but unfortunately present other drawbacks.

Another interesting choice is $F1$, the harmonic mean of Precision and Recall. It represents an compromise between both metrics, being a good choice even for very unbalanced datasets, and takes values between 1 for a perfect model and 0 for a model where Precision or Recall is 0. This metric can be generalized for multiclass problems in several ways, perhaps the most common being to calculate it for each label separately and finding their average weighted by support.

AUC, the Area Under the ROC Curve, is a less common score that is, nevertheless, a much better choice than Accuracy in general. [21] presents a revealing analysis of the properties of Accuracy and AUC, and concludes that AUC is consistent with Accuracy but more discriminative, and so a better metric overall. It has been used as the reference metric in [22] in a generalized version for multiclass classification. AUC



Figure 1.2.4: CRISP-DM diagram from https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining.

is, however, a not very popular metric, probably because it requires the classifier to yield scores, which is not always possible.

1.2.3 Standard Production Methodologies

Finally, several standards exist related to the process of developing a predictive model, with *Knowledge Discovery in Databases* (KDD), *Sample, Explore, Modify, Model and Assess* (SEMMA) and *Cross-industry Standard Process for Data Mining* (CRISP-DM) [23] being the most widespread. CRISP-DM is possibly the most popular, and has been extended to produce the new *Analytics Solutions Unified Method for Data Mining/Predictive Analytics* (ASUM-DM) [24] standard. These methodologies focus on the iterative nature of the predictive modeling process, and cover almost any data-driven application.

While CRISP-DM and ASUM-DM cover the whole predictive modeling cycle, including steps as business understanding or deployment of the built solution, KDD and SEMMA are focused on its core, ranging from data preparation to evaluation of the solution. The rest of this thesis is devoted to just a small part of the process: the modeling phase, including transformation of the data, training of a predictive model, and assessment of its performance.

1.3 Contributions

Artificial Neural Networks (ANNs) are some of the most powerful tools Machine Learning can provide nowadays for regression or classification problems. In a sense, they are just a generalization of linear models like Linear Regression and Logistic Regression, but their capabilities are much wider. Being universal approximators and having great flexibility to be adapted to many types of data makes ANNs a very promising tool for Machine Learning practitioners.

This thesis is structured in three main contributions related to ANNs. The first of them is an application of ANNs, and more precisely convolutional neural networks, to the problem of prediction in renewable energies. Data provided by Numerical Weather Prediction (NWP) [25] systems usually has a spatial structure that can be exploited by such models, resulting in an improvement of the results while keeping computational costs under control thanks to the efficiency of artificial neural networks.

In the second place, feature transformation algorithms can be used as a first step in the modelization pipeline for a Machine Learning problem, decomposing it in simpler subproblems for deeper understanding and a higher control on the costs of the solution. However, traditional feature transformation techniques are usually linear or kernel methods with analytical solutions either with limited expressive power or with a very high computational cost. This is the case of Fisher Discriminant Analysis (FDA), a feature transformation algorithm for classification problems. The contribution consists in exploiting an iterative alternative to traditional, eigenvalue-based, FDA training algorithm. This algorithm is based on Least-Squares and the solution produced is in some sense equivalent to that of traditional FDA, but its computational cost is much lower when working with *big data* problems, and, moreover, a non-linear extension based on ANNs can be applied instead of the usual kernel alternative, resulting in an efficient non-linear feature transformation that maximizes separability. Also, the performance of such ANNs in imbalanced classification problems will be studied.

In the third place, margin loss functions employed in support vector machines (SVMs) make models more stable and could be an interesting improvement for ANNs, that have traditionally used mean squared error in regression and cross-entropy in classification. The contribution is a comprehensive study on the performance of ANNs paired with ϵ -insensitive and hinge losses, and the effect of combining several losses in a single neural network architecture taking advantage of the flexibility of modern computation frameworks.

1.4 Structure

The structure of this thesis, with a brief summary of each of the chapters, is:

- **Chapter 1: Introduction**, the present chapter, contains a brief introduction to the field of Machine Learning, assuming the reader has basic knowledge in linear algebra, probability and optimization. The motivation and contributions are also detailed here.

- **Chapter 2: Machine Learning Core** describes linear models with direct training like Linear Regression, Ridge and Fisher's Discriminant Analysis, gives a brief description of Linear Models with iterative training, including Linear and Logistic Regression and Linear SVM. Also introduces a kernel extension with direct training for linear models, Kernel Ridge, and Kernel Discriminant Analysis, gives a brief description of Kernel Models with iterative training, Kernel SVMs, and concludes with an introduction to Artificial Neural Networks and Deep Learning.
- **Chapter 3: Prediction in Wind and Solar Energy** contains the contributions to the problem of prediction in Wind and Solar Energy published in [26]. The related publication [27] is also included.
- **Chapter 4: Deep Fisher Discriminant Analysis** contains the contributions to the generalization of Fisher Discriminant Analysis published in [28]. Related publications [29, 22] are included alongside the main contribution [28].
- **Chapter 5: Margin in Artificial Neural Networks** contains the contributions to the use of margin loss functions in Artificial Neural Networks published in [30]. Additional related publications [31, 32] are also included.
- **Chapter 7: Conclusion** finishes the main part of the thesis with a summary and a discussion on the results of the previous chapters and some hints for further work.
- **Appendix A: Publications** lists the articles published in journals and conferences during the elaboration of this thesis.
- **Appendix B: Software** lists the open source software repositories developed for the realization of the experiments in the associated publications.

Chapter 2

Machine Learning Core

The purpose of this chapter is to briefly introduce some predictive models that will make up the basis of the contributions and experimental results. The first objective is to present linear models with closed-form analytical solution. Once this first objective is achieved, a second goal is to introduce iterative training algorithms for those linear models that can make use of them. Then kernel methods will be presented to overcome the limitations of linear models by allowing non-linear transformations using the kernel trick. The last objective is to present Artificial Neural Networks, giving a brief summary of their strengths and weaknesses, as a non-linear alternative for Big Data setups.

The chapter is organized in three sections. The first one is focused on linear models and consists on three subsections. The first subsection is a brief introduction to basic linear models for regression problems that have analytic solutions, covering Linear and Ridge Regression, and Fisher's Discriminant Analysis, while the second subsection introduces linear models with iterative training algorithms resulting in what is known as Generalized Linear Models. The third subsection introduces the concept of margin, which results in the well known Linear Support Vector Machines. The second section introduces kernel models and is also structured in two subsections, the first one dedicated to Kernel Ridge and Kernel Discriminant Analysis and the second one focused on Kernel Support Vector Machines. The third section is devoted to Deep Models, with subsections on approximation (Multilayer Perceptrons, Convolutional Neural Networks, etc.) and optimization (Backpropagation, overfit, underfit, etc.).

2.1 Linear Models

Focusing on parametric models, and according to the Occam's Razor, the model with the smallest number of parameters should be considered first when facing a Machine Learning problem. Therefore, linear models are usually a natural starting point.

Linear models produce outputs that are linear combination of the input features. Therefore, their parametric structure consists simply on a numerical vector $w \in \mathbb{R}^d$ and a *bias* term $b \in \mathbb{R}$, also called *intercept*:

$$\hat{y} = m(x) = x^t w + b.$$

For convenience, it is usual to prepend a 1s column to the input matrix X , resulting in a matrix of size $N \times (d+1)$, so the bias can be prepended to vector w for all the model parameters to be handled jointly. Also, this notation can accommodate multivariate targets. It is usual to denote the parameter matrix as W , and the model becomes

$$\hat{Y} = m(X) = XW.$$

Some of those models, like *Linear Regression* for regression tasks, *Principal Component Analysis* for unsupervised feature tranformation and *Linear Discriminant Analysis* for feature transformation in classification setups, can be fitted with a direct algorithm that provides a closed-form analytical solution. However, no direct algorithm is available for most models, that have to be trained with a family of algorithms completely different from the direct ones mentioned previously. In fact, this kind of iterative algorithms can fit linear models that don't have an analytic, closed-form solution. Although an analytic solution might seem the best option at first sight, the high computational cost usually associated to such solutions can become a daunting problem when dealing with big datasets. Iterative training algorithms can only provide numerical solutions that approximate the analytic solution, but on the other hand their computational cost is usually smaller and can be controlled easily by limiting the number of iterations.

2.1.1 Linear and Ridge Regression

If the problem at hand is a regression problem, that is, the target Y is available for the training dataset and it is numerical, *Linear Regression* is the most basic model suitable for the task.

There are many acceptable definitions of the straight line that best fits the data, but it is usual to look for the one that minimizes the sum of the squares of the residuals, that is, the squares of the differences between targets and predictions [15, 13]. This objective function to be minimized is the aforementioned RSS

$$L(Y, \hat{Y}) = \|Y - \hat{Y}\|^2 \tag{2.1.1}$$

and, since it is a convex and differentiable function, its optimization is quite simple:

$$\frac{\partial L}{\partial W} = -2X^t(Y - \hat{Y}) = -2X^t(Y - XW) = 0 \tag{2.1.2}$$

$$\iff X^t(Y - XW) = 0 \tag{2.1.3}$$

$$\iff W^* = (X^tX)^{-1}X^tY = S^{-1}X^tY, \tag{2.1.4}$$

where $S = X^tX$ is the empirical covariance matrix.

This method is known as *Ordinary Least Squares* (OLS), and, as it has been outlined, it is a direct method that provides an analytical solution to the problem.

Recall N is the number of observations in the dataset, d is the dimension of the input space and d' is the dimension of the target space. Then, solving 2.1.4 has a cost of

1. $\mathcal{O}(d^2N)$ to compute X^tX .

2. $\mathcal{O}(d^3)$ to compute its inverse.
3. $\mathcal{O}(d'dN)$ to compute X^tY .
4. $\mathcal{O}(d'd^2)$ to compute the product of the previous two.

$\mathcal{O}(d^2N)$ and $\mathcal{O}(d'dN)$ are the dominant terms when the size of the dataset is greater than the number of features or the dimension of the target, while $\mathcal{O}(d^3)$ or $\mathcal{O}(d'd^2)$ dominate otherwise.

Some assumptions have to be made on the data to get a good model:

Linearity: the response variable is a linear combination of the independent variables.

Absence of multicollinearity: no collinearity between the predictor variables exists.

Homoscedasticity: the variance of the residues (errors) is constant for different values of the predictor variables.

No autocorrelation of the errors: the errors are uncorrelated between observations.

Naturally, if these assumptions are not fulfilled, the model will yield poor results, or even worse, it will not be applicable. This is the case when the assumption of *absence of multicollinearity* is not met, which leads to a non-invertible matrix X^tX that precludes the application of OLS. Also, if the number of features is higher than the size of the training dataset, the model can easily overfit, yielding poor results.

All this can be mitigated by adding a \mathcal{L}^2 regularization term to 2.1.1

$$L(Y, \hat{Y}) = \|Y - \hat{Y}\|^2 + \alpha \|W\|_2^2, \quad (2.1.5)$$

whose analytical solution is

$$W = (S + \alpha I_d)^{-1} X^t Y = D_\alpha X^t Y, \quad (2.1.6)$$

with

$$D_\alpha = (S + \alpha I_d)^{-1}.$$

The Linear Regression model with \mathcal{L}^2 regularization is known as *Ridge Regression*.

2.1.2 Fisher's Linear Discriminant Analysis

While several unsupervised feature transformation techniques exist, like the very popular Principal Component Analysis (PCA), the application of such tools focused on describing data is not guaranteed to perform well in a classification setup. Fisher's Linear Discriminant Analysis, on the other hand, is a linear feature transformation technique designed specifically for classification problems. It intends to reduce the dimensionality of the data while maximizing the *separability* of the different classes in the projected features, that are linear combinations of the original ones. This goal makes possible its application in classification problems. There are several definitions of *separability* that can be used, but probably the most commonly used function for maximization is

$$g(W) = \text{trace}(s_T^{-1} s_B) = \text{trace}((A^t S_T A)^{-1} (A^t S_B A)), \quad (2.1.7)$$

where A is the $d \times (c - 1)$, with c the number of classes, projection matrix, $S_B = X^t C E \Pi^{-1} E^t C X$ and s_B denote the between-class covariance matrices of the original data and the projections $z = Ax$, respectively, and $S_T = X^t C X$ and s_T denote the total covariance matrices of the original data and the projections, respectively [33]. Here Π denotes the $c \times c$ diagonal matrix with $\Pi_{nn} = N_n$ the number of sample patterns in class n , E denotes the $N \times c$ one-hot encoding matrix, and $C \in \mathbb{R}^{N \times N}$ the centering matrix

$$C = I_N - \frac{1}{N} \mathbf{1}_N \mathbf{1}_N^t,$$

with $\mathbf{1}_N$ the all ones N -dimensional vector.

Solving $\nabla_A g = 0$ leads to

$$0 = -2S_T A s_T^{-1} s_B s_T^{-1} + 2S_B A s_T^{-1} \quad (2.1.8)$$

and, hence, to

$$S_T^{-1} S_B A = A s_T^{-1} s_B, \quad (2.1.9)$$

that, up to an invertible transformation of A with no effect on the cost function $g(W)$, gives the eigenvalue problem

$$S_T^{-1} S_B A = A \Lambda, \quad (2.1.10)$$

with Λ the non-zero eigenvalues of $S_T^{-1} S_B$.

Therefore, maximizing $g(A)$ will be achieved by sorting the eigenvalues of Λ and selecting the $q = \min\{d, c - 1\}$ eigenvectors associated to the largest ones, with c being the number of classes. The solution is not unique and it is common to impose the normalization $A^t S_T A = I_q$. Also, in some problems S_T may not be invertible and it is frequent to use the regularized version $S_T + \lambda I$ with an appropriate $\lambda > 0$ instead.

Solving the eigenvalue problem would require the computation of

1. S_T at a cost of $\mathcal{O}(Nd^2)$.
2. Its inverse at a cost of $\mathcal{O}(d^3)$.
3. $S_T^{-1} S_B$ at a cost of $\mathcal{O}(d^3)$.
4. Its SVD decomposition at a cost of $\mathcal{O}(d^3)$

Therefore three steps with a cost $\mathcal{O}(d^3)$ are needed.

2.1.3 Iterative Linear Model Training

2.1.3.1 Linear and Ridge Regression

OLS will produce an exact, analytical solution for linear regression, and the same is true for the direct training of ridge regression. However, direct methods present a series of important drawbacks.

First, consider the possibility that new, labelled data would be made available. This would potentially lead to a solution of higher quality through a lower estimation

error, the one related to the quality of the data, but it is not simple to update the parameters of a trained model using the new data with OLS, that consists on a single step calculation over the full data matrices X and Y . Also, the computational cost of OLS can be too high since the cost is not linear in terms of d .

Those drawbacks could be prevented with a different kind of algorithm, in particular an iterative algorithm based on *gradient descent*. The idea behind such methods is that the data is processed sequentially in *batches*, starting in a random initial configuration of the model that is refined with each iteration by using a new data batch to push the parameters in the direction with the highest decrease of the loss function.

Such algorithms can't produce a perfect, analytical solution such as the direct ones, but on the other hand the learning is incremental and the full data matrix is never used unless the batch size, that is a free hyper-parameter, is equal to the dataset size. In the case of linear regression, this is done by minimizing the residual sum of squares

$$\begin{aligned} E &= \|Y - \hat{Y}\|^2 \\ &= \|Y - XW\|^2, \end{aligned}$$

for batches of N data. Gradient descent consists on updating the parameters in the opposite direction of the gradient of the loss with respect to W , whose value is

$$\nabla E(W) = -2X^t(Y - XW),$$

which points in the direction of steepest ascent of the loss function, and, therefore, opposite to the desired direction for the parameter updated:

$$W \leftarrow W - \eta \nabla E(W), \tag{2.1.11}$$

where $\eta \in (0, \infty)$ is the *learning rate*, a hyper-parameter that determines the magnitude of the parameter updates.

Gradient descent has some drawbacks in addition to the lack of a perfect analytical fit of the model's parameters, particularly the sensibility to the learning rate and to the disparity in the scale of the variables. It is usual to standardize variables prior to the application of the gradient descent algorithm to ease these problems.

2.1.3.2 Logistic Regression

So far, all linear models discussed handle numerical features and responses, but categorical variables are common in real-world applications, both as input or as target. Some models, such as those based on *decision trees*, can work with categorical data out of the box, but linear models and derivatives need such variables to be transformed into numerical values. The most common transformation for categorical features is *one-hot encoding*, consisting in the substitution of the original categorical features by dummy *one-out-of- K* features, but there are many other alternatives. For categorical targets, that is, the target of a classification problem, one-hot encoding is always used, with one of the dummy variables generated being dropped when the problem is binary.

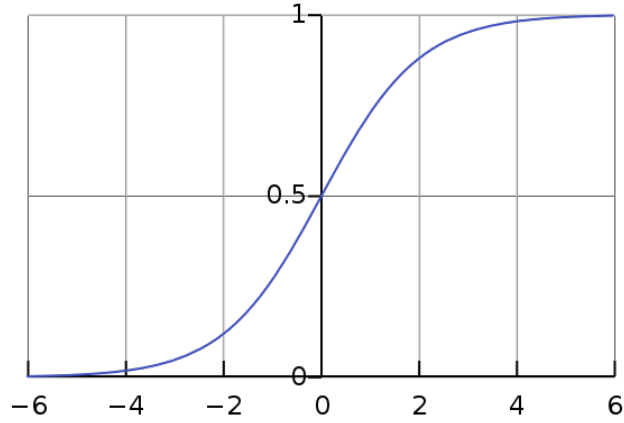


Figure 2.1.1: Logistic function

Considering a classification problem, and assuming all the categorical data has been properly encoded, nothing prevents linear regression from being used, but performance will often be poor. It is advisable to adapt the model for the kind of task at hand, and this consists in modulating the output by a *logistic* function,

$$P(\hat{Y}|X) = \frac{1}{1 + e^{-Z}}, \quad (2.1.12)$$

where $Z = XW$ is the output before modulation, represented graphically in 2.1.1, or a *softmax* [13] function,

$$P(\hat{Y}_c|X) = \frac{e^{Z_c}}{\sum_{i=1}^C e^{Z_i}}, c = 1 \dots C, \quad (2.1.13)$$

where $Z_c = XW_c$ is the output corresponding to class c before modulation and W_c the set of parameters corresponding to class c , so it resembles the one-hot encoded target and can be directly interpreted as a conditional probability.

Also, given the particular form of the target, the squared error loss function is inappropriate [34], and *cross-entropy*,

$$E = - \sum_{n=1}^N -Y_n \log(Z_n) - (1 - Y_n) \log(1 - Z_n), \quad (2.1.14)$$

with $Z_n = X_n W$, in the case of binary classification, or *categorical cross-entropy*

$$E = - \sum_{n=1}^N \sum_{c=1}^C Y_{n,c} \log(Z_{n,c}), \quad (2.1.15)$$

with $Z_{n,c} = X_n W_c$, in the case of multiclass classification, is a more recommendable choice.

Observing the loss function, it becomes clear that a direct method to get an exact analytical solution of the logistic or multinomial regression is not available, so in this case resorting to an iterative algorithm, such as SGD, is a need.

2.1.4 Linear Support Vector Machines

The training of a linear SVM model can be performed with a variety of iterative algorithms. SGD is sometimes used, taking the appropriate precautions to deal with the non-differentiability of the loss function, when the number of samples in the training dataset is bigger than the number of features, while *Coordinate Descent* [35] is usually preferred in the opposite case.

2.1.4.1 Linear Support Vector Classifiers

Let's focus on a binary classification problem. It is said to be *linearly separable* if a linear model m such that $m(x_i) = y_i, y_i \in \{+1, -1\} \quad \forall i = 1 \dots N$ can be found. If a problem is linearly separable, an infinite number of linear solutions will exist, but it is clear that not all of them have the same generalization capability. Support Vector Machines have the goal of finding the maximum margin solution of a binary classification problem, that is, the separating plane that maximizes the distance to the points of the data sample or *margin*, so that if a sample is slightly modified, it will be still correctly classified.

If the training dataset is defined by

$$\{(x_1, y_1) \dots (x_N, y_N)\}, x_n \in \mathbb{R}^d, y_n \in \{-1, 1\},$$

and assuming the problem is linearly separable, that is, W, b exist such that

$$y_n(W^t x_n + b) > 0, n = 1 \dots N,$$

then the margin of a separating plane is

$$\rho = \min_{x_n} \frac{1}{\|W\|} |W^t x_n + b|.$$

The optimal plane is then the W that solves

$$\begin{aligned} & \operatorname{argmax}_{W,b} \quad \rho \\ & \text{s.t.} \quad \frac{1}{\|W\|} y_n(W^t x_n + b) \geq \rho, n = 1 \dots N, \end{aligned}$$

and, taking $\|W\| = \frac{1}{\rho}$, it becomes

$$\begin{aligned} & \operatorname{argmax}_{W,b} \quad \frac{1}{\|W\|} \\ & \text{s.t.} \quad y_n(W^t x_n + b) \geq 1, n = 1 \dots N, \end{aligned}$$

which is equivalent to

$$\begin{aligned} & \operatorname{argmin}_{W,b} \quad \frac{1}{2} \|W\|^2 \\ & \text{s.t.} \quad y_n(W^t x_n + b) \geq 1, n = 1 \dots N, \end{aligned}$$

that is more convenient to work with.

The assumption of separability is too strong in general, but some flexibility can be induced with slack variables ξ_n

$$\begin{aligned} \operatorname{argmin}_{W,b,\xi} \quad & C \sum_{n=1}^N \xi_n + \frac{1}{2} \|W\|^2 \\ \text{s.t.} \quad & \begin{cases} y_n(W^t x_n + b) \geq 1 - \xi_n, n = 1 \dots N, \\ \xi_n \geq 0, n = 1 \dots N, \end{cases} \end{aligned} \quad (2.1.16)$$

or, equivalently,

$$\begin{aligned} \operatorname{argmin}_{W,b,\xi} \quad & C \sum_{n=1}^N \xi_n + \frac{1}{2} \|W\|^2 \\ \text{s.t.} \quad & \begin{cases} \xi_n \geq 1 - y_n(W^t x_n + b), n = 1 \dots N, \\ \xi_n \geq 0, n = 1 \dots N, \end{cases} \end{aligned}$$

where ξ_n are the *slack variables* and C is the hyper-parameter that controls the slacking allowed on the loss function. A point x_n will have an associated slack variable $\xi_n = 0$ if it is correctly classified and further from the separating hyperplane than the margin, $0 < \xi_n \leq 1$ if correctly classified but not far enough from the hyperplane, and $\xi_n > 1$ if missclassified.

This constrained optimization problem can be reformulated to incorporate the constraints as part of the objective function,

$$\operatorname{argmin}_{W,b} \quad C \sum_{n=1}^N \max(0, 1 - y_n(W^t x_n + b)) + \frac{1}{2} \|W\|^2.$$

This problem is equivalent to that of a linear model with Tikhonov regularization minimizing the *hinge loss*,

$$\sum_{n=1}^N \max(0, 1 - y_n(W^t x_n + b)) + \alpha \|W\|^2.$$

Once the Linear Support Vector Classifier has been fitted, predictions can be obtained with

$$\hat{y} = \operatorname{sign}(W^t x + b).$$

As will be shown later, obtaining a formulation of the model in terms of the inner products of the data can be of interest. To do so, let's start by writing down the Lagrangian of (2.1.16) which is

$$L = C \sum_{n=1}^N \xi_n + \frac{1}{2} \|W\|^2 - \sum_{n=1}^N \alpha_n (y_n(W^t x_n + b) - (1 - \xi_n)) - \sum_{n=1}^N \mu_n \xi_n, \quad (2.1.17)$$

where $\alpha_n, \mu_n \geq 0 \quad \forall n$, that, minimizing w.r.t W , b and ξ_n , that is, making zero the respective derivatives, gives

$$\begin{aligned} W &= \sum_{n=1}^N \alpha_n y_n x_n, \\ 0 &= \sum_{n=1}^N \alpha_n y_n, \\ \alpha_n &= C - \mu_n, \end{aligned} \tag{2.1.18}$$

and substituting (2.1.18) into (2.1.17), the dual formulation of the problem is obtained:

$$\begin{aligned} D(\alpha) &= \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{n'=1}^N \alpha_n \alpha_{n'} y_n y_{n'} x_n^t x_{n'} \\ s.t. &\begin{cases} \sum_{n=1}^N \alpha_n y_n = 0, \\ 0 \leq \alpha_n \leq C, n = 1 \dots N. \end{cases} \end{aligned} \tag{2.1.19}$$

After solving the dual (usually with SMO [36, 37]), the Karush-Kuhn-Tucker (KKT) conditions can be applied on the support vectors SV to get the optimal b and the predictor is given by

$$\hat{y} = \text{sign}\left(\sum_{n \in SV} y_n \alpha_n x_n^t x + b\right).$$

The idea of improving the linear model for classification with the concept of margin is now easy to implement, just by replacing the traditional cross-entropy loss by the hinge loss. A k -class problem with $k > 1$ can be reformulated as k 1-versus-rest binary classification subproblems, and the binary SVM can be used as the classifier for each of the resulting subproblems.

2.1.4.2 Linear Support Vector Regressors

The same idea is applicable to regression problems. Consider now the training dataset as

$$\{(x_1, y_1) \dots (x_N, y_N)\}, x_n \in \mathbb{R}^d, y_n \in \mathbb{R}.$$

The idea now is to define a tolerance ϵ and minimize the error given by

$$\begin{cases} -(\hat{y} - y) - \epsilon & \hat{y} - y < -\epsilon \\ 0 & -\epsilon \leq \hat{y} - y \leq \epsilon \\ (\hat{y} - y) - \epsilon & \epsilon < \hat{y} - y, \end{cases}$$

where it is clear that differences between y and \hat{y} smaller than ϵ are not penalized.

The regression plane produced will keep the training data at a distance smaller than ϵ , which, again, might be a too strong assumption that can be relaxed using

slack variables ξ_n to arrive to the optimization problem

$$\begin{aligned} \operatorname{argmin}_{W,b,\xi,\tilde{\xi}} \quad & C \sum_{n=1}^N (\xi_n + \tilde{\xi}_n) + \frac{1}{2} \|W\|^2 \\ \text{s.t.} \quad & \begin{cases} \xi_n \geq -(y_n - (W^t x_n + b)) - \epsilon, n = 1 \dots N, \\ \tilde{\xi}_n \geq y_n - (W^t x_n + b) - \epsilon, n = 1 \dots N, \\ \xi_n, \tilde{\xi}_n \geq 0, n = 1 \dots N. \end{cases} \end{aligned} \quad (2.1.20)$$

This can be reformulated as

$$\operatorname{argmin}_{W,b} \quad C \sum_{n=1}^N \max(0, |y_n - (W^t x_n + b)| - \epsilon) + \frac{1}{2} \|W\|^2,$$

which is equivalent to the problem of a linear model with Tikhonov regularization minimizing the *epsilon-insensitive* loss,

$$\sum_{n=1}^N \max(0, |y_n - (W^t x_n + b)| - \epsilon) + \alpha \|W\|^2.$$

Predictions for a fitted Linear Support Vector Regressor can be obtained with

$$\hat{y} = W^t x + b.$$

Again, a dual formulation of the problem can be obtained. The Lagrangian of (2.1.20) is

$$\begin{aligned} L = \quad & C \sum_{n=1}^N (\xi_n + \tilde{\xi}_n) + \frac{1}{2} \|W\|^2 - \sum_{n=1}^N \alpha_n (\xi_n + y_n - (W^t x_n + b) + \epsilon) \\ & - \sum_{n=1}^N \alpha_n^* (\tilde{\xi}_n - y_n + (W^t x_n + b) + \epsilon) - \sum_{n=1}^N \mu_n \xi_n - \sum_{n=1}^N \mu_n^* \tilde{\xi}_n, \end{aligned} \quad (2.1.21)$$

where $\alpha_n, \alpha_n^*, \mu_n, \mu_n^* \geq 0 \quad \forall n$. Minimizing it w.r.t W, b, ξ_n and $\tilde{\xi}_n$, that is, computing the partials and equating to 0 gives

$$\begin{aligned} W &= \sum_{n=1}^N (\alpha_n - \alpha_n^*) x_n, \\ 0 &= \sum_{n=1}^N (\alpha_n - \alpha_n^*), \\ \alpha_n &= C - \mu_n, \\ \alpha_n^* &= C - \mu_n^*, \end{aligned} \quad (2.1.22)$$

Substituting (2.1.22) into (2.1.21) gives the dual problem

$$\begin{aligned}
D(\alpha, \alpha^*) &= -\frac{1}{2} \sum_{n=1}^N \sum_{n'=1}^N (\alpha_n - \alpha_n^*) x_n^t x_{n'} (\alpha_{n'} - \alpha_{n'}^*) + & (2.1.23) \\
&\sum_{n=1}^N (\alpha_n - \alpha_n^*) y_n - \sum_{n=1}^N (\alpha_n + \alpha_n^*) \epsilon, \\
s.t. &\begin{cases} \sum_{n=1}^N \alpha_n y_n = 0, \\ 0 \leq \alpha_n, \alpha_n^* \leq C, n = 1 \dots N. \end{cases}
\end{aligned}$$

As in the classification case, the dual can be solved with SMO and the KKT conditions can be applied once again to arrive to

$$\hat{y} = \sum_{n \in SV} (\alpha_n - \alpha_n^*) x_n^t x + b.$$

2.2 Kernel Models

So far, all the models covered are linear. Linear models have many desirable properties such as simplicity, explainability and stability among others, but often lack the expressive power needed to solve real-world problems with the required accuracy.

Fortunately, many linear models can be altered to overcome some of their limitations with a slight, elegant modification: the kernel trick, sacrificing explainability to achieve a greater expressiveness in the process.

This technique allows to transform an algorithm expressed in terms of an internal product of the form $x^t x$ and generalize it with a non-linear kernel. In more detail, consider an expansion of x , $\tilde{x} = \phi(x)$, with ϕ a non-linear transformation from the original d -dimensional feature space to a new D -dimensional feature space, usually with $D \gg d$. The expanded data matrix will be denoted as \tilde{X} .

If the original algorithm in terms of X can be expressed in terms of the dot product XX^t , then this product can be substituted by the kernel matrix $K = \tilde{X}\tilde{X}^t$ and the algorithm will benefit from the D -dimensional representations without the need to explicitly compute them. Thus, a *kernel machine* is a linear model where this trick is used, replacing all appearances of XX^t by K without needing explicit knowledge of the transformation ϕ .

2.2.1 Kernel Ridge

The analytical solution of ridge regression, (2.1.6), can be rewritten as

$$W = (X^t X + \alpha I)^{-1} X^t Y = X^t (X X^t + \alpha I)^{-1} Y,$$

and, applying the kernel,

$$\tilde{W} = \tilde{X}^t (K + \alpha I)^{-1} Y.$$

Computing \tilde{W} is unfeasible due to the dimension of \tilde{X} , but in practice only $(K + \alpha I)^{-1} Y$ will be needed to fit the model and perform inference, at a cost of

1. $\mathcal{O}(N^2\kappa)$, where κ is the cost of a kernel computation $k(x, x')$, to compute K .
2. $\mathcal{O}(N^3)$ to compute the inverse.
3. $\mathcal{O}(d'N^2)$ to compute the final product with Y .

For inference, the predictions are given by

$$\hat{Y}_{test} = \tilde{X}_{test} \tilde{W} = \tilde{X}_{test} \tilde{X}^t (K + \alpha I)^{-1} Y = K_{test} (K + \alpha I)^{-1} Y,$$

where $X_{test} = \{x'_1 \dots x'_M\}$ is the matrix with M unseen, test data, and $K_{test} = (k'_1 \dots k'_M)^t$, with $k'_m = (k(x'_m, x_1) \dots k(x'_m, x_N))$. These predictions have a computational cost of

1. $\mathcal{O}(NM\kappa)$, where κ is the cost of a kernel computation $k(x, x')$, to compute K_{test} .
2. $\mathcal{O}(d'MN)$ to compute the final product.

2.2.2 Kernel Fisher's Discriminant Analysis

The kernel-based extension of Fisher's Discriminant Analysis [38, 39] is also quite popular. Mapping X into a new feature space with a non-linear transformation and computing Discriminant Analysis there will provide the flexibility needed in many real world problems. Several formulations of the method, like that of [40], have been proposed. The one published in [28] will be followed here.

If \tilde{X} is again the projected data, then the optimal kernel discriminant will be given by the W that maximizes

$$g(W) = \text{trace}(s_T^{-1} s_B) = \text{trace} \left((A^t \tilde{S}_T A)^{-1} (A^t \tilde{S}_B A) \right), \quad (2.2.1)$$

where the notation is the same as in LDA but for $\tilde{S}_B = \tilde{X}^t C E \Pi^{-1} E^t C \tilde{X}$ and $\tilde{S}_T = \tilde{X}^t C \tilde{X}$ that denote the between-class and total covariance matrices of the expansions \tilde{X} .

Again, this leads to the eigenvalue problem

$$\tilde{S}_T^{-1} \tilde{S}_B \tilde{A} = \tilde{A} \Lambda, \quad (2.2.2)$$

with Λ the non-zero eigenvalues of $\tilde{S}_T^{-1} \tilde{S}_B$. But in this case computations cannot be performed in general in the high-dimensional space, so a reformulation is needed to apply the kernel trick. We discuss it next.

Since \tilde{S}_T might not be invertible, a regularized version of the problem like

$$\tilde{D}_\lambda \tilde{S}_B \tilde{A} = \tilde{A} \Lambda, \quad (2.2.3)$$

where $\tilde{D}_\lambda = (\tilde{S}_T + \lambda I)^{-1}$, might be needed. Let's set $\tilde{Q} = \tilde{X}^t C E \Pi^{-1/2}$ and define the matrix R as

$$R = \tilde{Q}^t \tilde{D}_\lambda \tilde{Q} = \Pi^{-1/2} E^t C \tilde{X} \tilde{D}_\lambda \tilde{X}^t C E \Pi^{-1/2}, \quad (2.2.4)$$

then its SVD decomposition is $R = V\Lambda V^t$ and, since the pair $(\tilde{A} = \tilde{D}_\lambda \tilde{Q}V, \Lambda)$ verifies

$$\begin{aligned}\tilde{D}_\lambda \tilde{S}_B \tilde{A} &= \tilde{D}_\lambda \tilde{Q} \tilde{Q}^t \tilde{D}_\lambda \tilde{Q}V \\ &= \tilde{D}_\lambda \tilde{Q}RV \\ &= \tilde{D}_\lambda \tilde{Q}V\Lambda,\end{aligned}$$

the projection matrix $\tilde{A} = \tilde{D}_\lambda \tilde{Q}V$ is a valid solution.

Let's now rewrite $\tilde{D}_\lambda \tilde{X}^t C$ by setting $\Psi = C\tilde{X}\tilde{X}^t C = CKC$, and $\Delta_\lambda = (\Psi + \lambda I)^{-1}$. Then

$$\begin{aligned}\tilde{X}^t C &= \tilde{X}^t C(\Psi + \lambda I)\Delta_\lambda \\ &= (\tilde{X}^t C C \tilde{X} \tilde{X}^t C + \lambda \tilde{X}^t C)\Delta_\lambda \\ &= (\tilde{X}^t C C \tilde{X} + \lambda I)\tilde{X}^t C \Delta_\lambda \\ &= \tilde{D}_\lambda^{-1} \tilde{X}^t C \Delta_\lambda;\end{aligned}$$

therefore

$$\tilde{D}_\lambda \tilde{X}^t C = \tilde{X}^t C \Delta_\lambda, \quad (2.2.5)$$

which implies

$$\begin{aligned}R &= \Pi^{-1/2} E^t C \tilde{X} (\tilde{D}_\lambda \tilde{X}^t C E) \Pi^{-1/2} \\ &= \Pi^{-1/2} E^t C \tilde{X} (\tilde{X}^t C \Delta_\lambda) E \Pi^{-1/2} \\ &= \Pi^{-1/2} E^t (C \tilde{X} \tilde{X}^t C) \Delta_\lambda E \Pi^{-1/2} \\ &= \Pi^{-1/2} E^t \Psi \Delta_\lambda E \Pi^{-1/2}.\end{aligned} \quad (2.2.6)$$

So, only the kernel matrix is needed to calculate matrix R and to compute its SVD decomposition to get V and Λ . Using (2.2.5), \tilde{A} is

$$\begin{aligned}\tilde{A} &= \tilde{D}_\lambda \tilde{Q}V = \tilde{D}_\lambda \tilde{X}^t C E \Pi^{-1/2} V \\ &= \tilde{X}^t C \Delta_\lambda E \Pi^{-1/2} V.\end{aligned} \quad (2.2.7)$$

\tilde{A} seems to require \tilde{X} but this can be avoided when computing the projections \tilde{x} of a new x , that would be

$$\begin{aligned}z &= \tilde{A}^t (\tilde{x} - \tilde{m}) = V^t \Pi^{-1/2} E^t \Delta_\lambda C \tilde{X}^t (\tilde{x} - \tilde{m}) \\ &= V^t \Pi^{-1/2} E^t \Delta_\lambda C \left(k_x - \frac{1}{N} K \mathbf{1}_N \right),\end{aligned} \quad (2.2.8)$$

where k_x is the vector $(k(x, x_1), \dots, k(x, x_N))^t$ and only kernel operations are involved.

The computational cost for the training algorithm is:

- $\mathcal{O}(N^2 \kappa)$, where κ is the cost of a kernel computation $k(x, x')$, to compute K .
- $\mathcal{O}(N^3)$ to compute $\Delta_\lambda = (\Psi + \lambda I)^{-1}$.
- $\mathcal{O}(cN^2)$ to compute $E^t \Psi$ and $\Delta_\lambda E$.

- $\mathcal{O}(cN^2)$ to compute $R = \Pi^{-1/2}E^t\Psi\Delta_\lambda E\Pi^{-1/2}$.
- $\mathcal{O}(c^3)$ to compute the SVD decomposition of R to get V .

Since $N \gg c$, κ may be expected, the total cost is $\mathcal{O}(N^3)$.

The computational cost for inference is:

- $\mathcal{O}(NM\kappa)$ to compute $C(k_x - \frac{1}{N}K\mathbf{1}_N)$, where κ denotes the cost of a kernel computation $k(x, x')$.
- $\mathcal{O}(cN^2)$ to compute $V^t\Pi^{-1/2}E^t\Delta_\lambda$.
- $\mathcal{O}(cNM)$ to compute z .

Thus, assuming $c \geq \kappa$, the overall test cost is dominated by $\mathcal{O}(cN^2)$ or $\mathcal{O}(cNM)$.

2.2.3 Kernel Support Vector Machines

Finally, the changes needed for kernel SVMs will be discussed here.

2.2.3.1 Kernel SVC

The kernel extension of the SVC, one of the most popular classification models available for small and medium size datasets, is quite straightforward once the dual formulation of the problem is available.

The SVC is now expressed in terms of XX^t , and the kernel trick can be applied, transforming the model into a non-linear classifier replacing $x_n^t x_{n'}$ with the kernel function $k(x_n, x_{n'})$:

$$D(\alpha) = \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{n'=1}^N \alpha_n \alpha_{n'} y_n y_{n'} k(x_n, x_{n'}) \quad (2.2.9)$$

$$s.t. \begin{cases} \sum_{n=1}^N \alpha_n y_n = 0, \\ 0 \leq \alpha_n \leq C, n = 1 \dots N. \end{cases}$$

For the usual Gaussian kernel, it can be shown that there is a map ϕ projecting the original data into a reproducible kernel Hilbert space (RKHS) [41] such that $k(x_n, x_{n'}) = \phi(x_n) \cdot \phi(x_{n'})$. The final models will still be linear but no longer in the original features but on their kernel extensions $\phi(x)$. However, the exact knowledge of ϕ is not needed and can be implicitly handled.

After solving the dual with SMO, the KKT conditions can be applied again on the support vectors SV to get the optimal b and the predictor is given by

$$\hat{y} = \text{sign}\left(\sum_{n \in SV} y_n \alpha_n k(x_n, x) + b\right).$$

Notice again that only k is used for inference and no explicit knowledge on the extension function ϕ is needed.

2.2.3.2 Kernel SVR

Again, once the model is expressed in terms of XX^t , the kernel trick can be applied to arrive at

$$\begin{aligned}
 D(\alpha, \alpha^*) &= -\frac{1}{2} \sum_{n=1}^N \sum_{n'=1}^N (\alpha_n - \alpha_n^*) k(x_n, x_{n'}) (\alpha_{n'} - \alpha_{n'}^*) + \quad (2.2.10) \\
 &\quad \sum_{n=1}^N (\alpha_n - \alpha_n^*) y_n - \sum_{n=1}^N (\alpha_n + \alpha_n^*) \epsilon, \\
 \text{s.t. } &\begin{cases} \sum_{n=1}^N \alpha_n y_n = 0, \\ 0 \leq \alpha_n, \alpha_n^* \leq C, n = 1 \dots N. \end{cases}
 \end{aligned}$$

Once again, solve the dual is solved with SMO and the KKT conditions can be applied on the support vectors SV to get the optimal b . The predictor is given by

$$\hat{y} = \sum_{n \in SV} (\alpha_n - \alpha_n^*) k(x_n, x) + b$$

where only the kernel is needed.

2.2.3.3 Complexity and Sparsity of Kernel SVMs

The dual problem has to be solved in a space with dimension N . This involves having to handle the kernel matrix, with size $N \times N$, which implies a great computational cost for large samples. Moreover, the usual algorithm to solve the dual problem is Standard Sequential Minimization (SMO) which proceeds iteratively, with each iteration having a cost of $O(N)$.

Since the number of iterations is $\Omega(N_{SV})$, with N_{SV} the number of support vectors that is usually $\Theta(N)$, training will have a cost of $\Omega(N^2)$ or higher, often cubic in N . This will make SVM training infeasible if the dataset is big enough.

Also, the solution of the SVM is sparse, that is, the number of nonzero α_n^* values is less than N ; however, the number of support vectors (patterns x_n for which $\alpha_n^* > 0$) is often $\Theta(N)$, and so inference has a cost of $\Theta(N)$, making the model unusable in online settings.

2.3 Deep Models

Artificial Neural Networks (ANNs) are a non-linear generalization of the linear and logistic regression models described in previous sections. Aside from their non-linear capabilities, reflected in the fact that a neural network is a universal approximator, this kind of model, at least in its most common form of Multilayer Perceptron, is associated to an iterative training algorithm, Backpropagation, whose computational cost scales linearly with the size of the training dataset.

ANNs have had three key moments in their history. The first one was in the 50's, with the introduction of the McCulloch-Pitts artificial neuron [42] and the Rosenblatt's perceptron [43], which was trained in a supervised way with the algorithm of

the delta rule. This first period was followed by a long winter from the mid 60's to the mid 80's. The second started in the early 90s, motivated by the backpropagation algorithm [44] for the training of Multilayer Perceptrons (MLPs), where ANNs received a large interest that led to a very good understanding of neural networks with one or two layers and that established ANNs as the state of the art approach for classification and regression problems at the time. However, competing proposals (Kernel Support Vector Machines for *small* data and decision tree based ensembles for *big* data, mostly) and the inability to train *deep* MLPs, with three or more layers depending on the data and other factors, led to a declining interest in ANNs around the year 2000.

The third key moment for ANNs has come with Deep Neural Networks (DNNs) [45]. While they can be defined as just a simple enlargement of previously studied architectures, their enormous success shattering records in image, speech or video recognition, and other *perception* problems, has made them extremely popular and induced the development of very powerful software tools that make them accessible to a much wider audience.

The proposals by G. Hinton and R. Salakhutdinov [46] on the one hand, and by Y. Bengio and his coworkers [47] on the other, were crucial to overcome problems such as the vanishing gradient that had previously prevented the effective training of DNNs.

In more detail, the importance of weight initialization was finally noticed, and different successful proposals for this purpose were developed. The first proposal, pre-training MLPs by stacking of Restricted Boltzmann Machine (RBM) [46] or Autoencoder [47] layers adjusted in a greedy manner represented a great improvement on previous ANN models, and allowed the training of truly deep models for the first time. However, the high computational cost and sophistication of this method limited its popularity. The simpler initialization proposed in [34] yields comparable results without the complexity of stacked pre-training, and was soon accepted as the new standard.

Also, the effect of hidden layer activation functions was studied in depth [34], reaching the conclusion that functions that approach asymptotes more slowly were more convenient than classical sigmoid activation functions, the fact that such functions do not saturate so easily being crucial to avoid vanishing gradient.

Finally, new regularization procedures like dropout [48] allowed to control the high risk of overfitting present in very large neural networks with thousands or even millions of adjustable parameters.

New input processing schemes were next introduced, such as the rediscovered convolutional layers, first proposed by Y. LeCun in the late 1990's, or the Rectified Linear Unit (ReLU) activations [49], whose 0–1 derivatives lend stability to training. Training itself has also greatly changed and improved. The sensible (but modest) minibatch training has been complemented by new and powerful gradient descent techniques, such as Nesterov's variant of classical momentum [50], self adjusting learning rate procedures such as Adagrad [51], Adadelata [52] or Adam [53], or batch normalization techniques to control covariance shift [54, 55].

Alongside these theoretical contributions, very relevant technical advances were also made. First, new hardware architectures, more suitable for the era of Big Data,

were developed. In particular, the use of GPUs for scientific computation was a significant improvement where matrix–vector operations were performed, as was the case when training ANNs. Alternatives were made available when code vectorization was not possible, or as an addition to it, with distributed DNN training [56, 57, 58].

Second, the development of numeric and symbolic calculation frameworks [59, 60, 61, 58, 62, 63] that can automatically perform symbolic differentiation and yield efficient low level code to run in a variety of hardware, including GPUs, made the task of working with highly complex gradients, like those associated to general feedforward architectures, much simpler.

These two advances have resulted in a growing number of publicly available software platforms for the development of ANNs, either at a relative low level, such as Caffe [64] developed in C++, or a high level, relying in the numeric and symbolic frameworks mentioned before for the low level operations, like Pylearn2 [65] (paired with Theano [60]), whose development was later abandoned in favor of alternatives such as Keras [66] (running on top of Theano, Tensorflow, CNTK or MXNet [60, 58, 63, 62]) or Pytorch [67] (using Torch [59] as backend), all of them developed in Python [68].

The main consequence of all this has been an ongoing, tremendous research effort, with impressive results in a number of areas, particularly on computer vision, with extensive use of convolutional architectures inspired in the layer processing that takes place in the visual cortex [69] as a key factor, and speech recognition and natural language processing, where convolutional, recurrent and transformer architectures continue to be developed. In all these cases, the processing results in successively refined representations of input patterns that at the last hidden layer are powerful enough to be successfully exploited by simple readouts. In other words, DNN training can be seen as a particularly effective way to perform feature engineering, to the point that the field is often identified with representation learning [70].

2.3.1 Approximation

2.3.1.1 The Artificial Neuron

Linear models can be extended in other ways aside from kernels, and a natural source of inspiration for this is Neurobiology. At the end of the 19th century, Ramón y Cajal initiated the study of neurons, whose basic structure is shown in figure 2.3.1. Simplifying, the neuron operates by receiving electrical impulses coming from other neurons through the dendrites, aggregating and transforming them in the soma, and transmitting the result of that transformation through the axon.

In the middle of the 20th century, McCulloch and Pitts designed the artificial neuron trying to reproduce the capabilities of the biological neuron with an electronic circuit that can be seen in figure 2.3.2. Around 1955, Frank Rosenblatt used the artificial neuron to build the Perceptron, a first Machine Learning model. In this model, each input has an associated parameter or *weight*, and all the weights are collectively represented by matrix W :

$$\hat{Y} = XW.$$

which makes it equivalent to the linear model. The Heaviside *activation function*

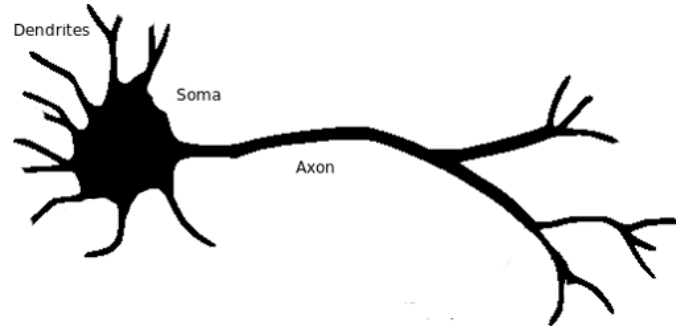


Figure 2.3.1: Neuron

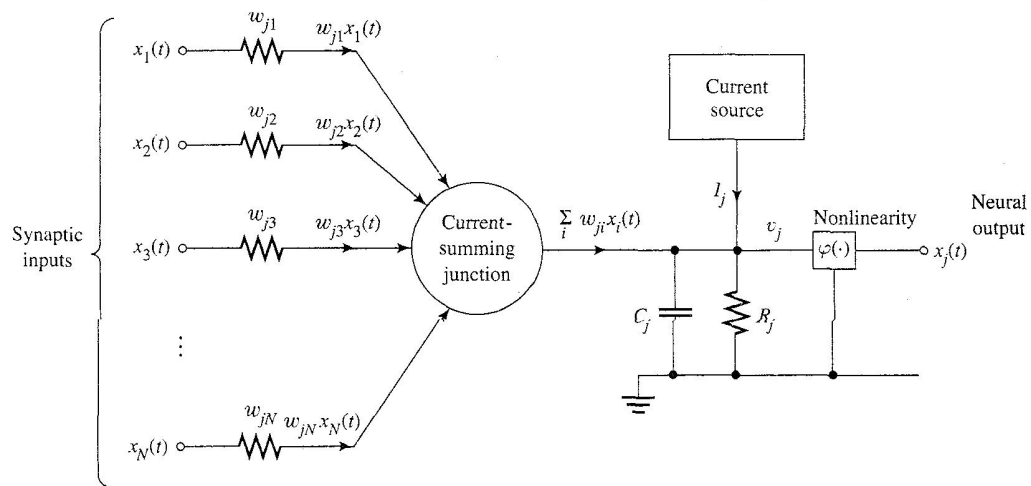


Figure 2.3.2: Artificial neuron

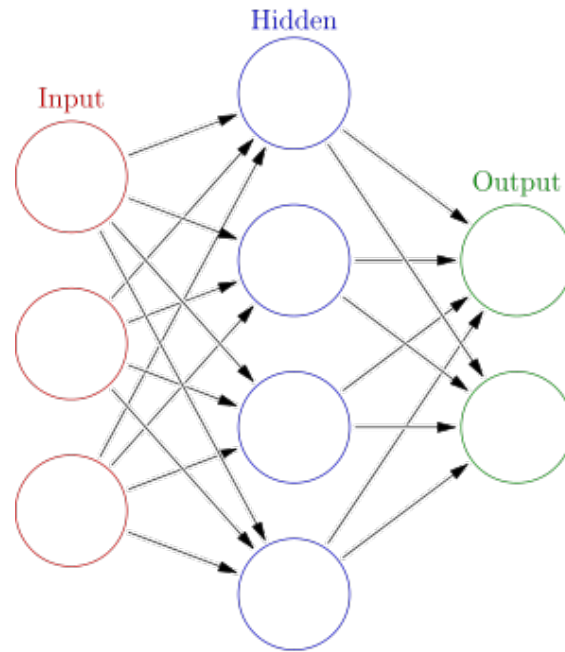


Figure 2.3.3: One-hidden-layer MLP, from Wikimedia Commons (distributed under a CC BY-SA 3.0 license)

was applied to the output, since the model was used for classification, thus making it quite similar to the Logistic Regression with logistic activation function.

So, the Perceptron works as a linear model, but the biological source of inspiration goes farther than that, because a typical nervous system has a huge number of neurons interconnected.

2.3.1.2 Representation Learning and Deep Learning: The Multilayer Perceptron

The usual way to organize neurons, from now on *units*, in an artificial neural network is in the form of layers, in what is known as a *feed-forward* architecture, or, more commonly, a *multilayer perceptron* (MLP). In this configuration, the input of the neural network is connected to the first of possibly several layers, called *hidden layers*. Each of them is then connected to the next one, and the last one is finally connected to the output layer which has a number of units equal to the dimension of the target. Every unit in one of these layers receives as input the linear combination of the outputs of the previous layer units, but no connections between units of the same layer or between units of non-consecutive layers exist, as outlined in figure 2.3.3.

By considering a single hidden layer, the mathematical formulation of the model becomes

$$\hat{Y} = f_2(f_1(XW_1)W_2),$$

where W_1 are the weights associated to the connections between the input and the hidden layer, and W_2 the ones associated to the connections between the hidden layer and the output layer. f_1 and f_2 are the activation functions corresponding to the

hidden and the output layer, respectively. f_2 in particular is determined by the type of the target. In a regression setting it will usually be the identity, while in a classification setting it will be a logistic or softmax function, depending on the number of classes. However, f_1 is a free hyper-parameter and can take many forms, with some caveats. If a linear function is used, since the function composition of linear functions is also linear, the model will remain linear. Therefore, to build a model that is more expressive than the linear or logistic regression, a non-linear activation function is needed. It was quite common to use the sigmoid or *tanh* function as hidden layer activation [71] until better options like rectified functions were explored [34, 72]. More hidden layers can be added to the model, resulting in the following expression for a two hidden layer network:

$$\hat{Y} = f_3(f_2(f_1(XW_1)W_2)W_3),$$

or a three hidden layer one:

$$\hat{Y} = f_4(f_3(f_2(f_1(XW_1)W_2)W_3)W_4).$$

MLPs are now considered *Representation Learning* models. Representation Learning is a branch of Machine Learning that comprises models that generate internal representations of the data that are more sophisticated and useful than the original one. When not one but several internal representations are generated, and each representation is built from the previous one increasing its level of abstraction, the model is said to be a *Deep Learning* model. Each hidden layer of a MLP generates one of such representations, and the level of abstraction and complexity grows with each additional layer until the output layer is reached, where the regression or classification problem is solved.

Theoretical results by Cybenko [73] and Hornik [74] indicate that MLPs are, in fact, universal approximators:

Theorem 1 (Universal Approximation Theorem). *Let ϕ be a continuous, non-constant, bounded and monotonic increasing function. Let $A \subseteq \mathbb{R}^m$ be compact, and $C(A)$ the space of all continuous functions in A . Then, $\forall \epsilon > 0$ and $\forall f \in C(A)$, $\exists N \in \mathbb{N}$, $v_i, b_i \in \mathbb{R}$, $w_i \in \mathbb{R}^m$ with $i = 1, \dots, N$, and a function*

$$F(X) = \sum_{i=1, \dots, N} v_i \phi(w_i^t x + b_i)$$

can be defined such that

$$|F(X) - f(x)| < \epsilon, \forall x \in A.$$

That is, a single non-linear hidden-layer MLP that can approximate any given *reasonable* function exists, although the number of units needed in the hidden layer remains undetermined.

Adding layers can increase the model expressiveness and performance [75] even if the number of units per layer decreases.

2.3.1.3 Spatial data and structured features: The Convolutional Neural Network

Artificial neural networks, and multilayer perceptrons in particular, are quite flexible models, and can be modified in order to take advantage of the particular properties of a given dataset, which will usually result in a lower approximation error. For example, some architectural adaptations have been developed to deal with “perception” data, such as images. This kind of data implies a topology between the variables, a grid structure present in the columns of each row of X . If such variables are rearranged by swapping columns in X , for example, then the image will lose its sense.

A typical image has a number of pixels organized as rows and columns, but also channels, which can be understood as analogous to features, usually representing color as grayscale or RGB. So, the number of channels in the data will usually be either one or three:

- If the input data is a grayscale image, then it will have a single channel.
- If the input data is a color image, then it will usually have three (R, G and B) channels.

However, if the input data is not an image but another type of grid [27, 26] then the number of channels might be different.

Classical Machine Learning models, like linear models, decision trees, SVMs or standard MLPs are not capable of taking advantage of this feature structure without the help of feature engineering [76, 77, 78]. That is, for one of such models, if a set of images is transformed by applying a particular random permutation of the pixels (always the same permutation to every picture on the dataset) the model performance will be roughly the same as with the original dataset: the model is not aware of the importance of the grid structure of the variables.

Also, while traditional MLP architectures usually benefit from the use of a high number of units in the hidden layers, this leads to a high number of weights, $M \times M'$ if an M unit layer is connected to an M' unit one, a number that can become rather large if, for instance, inputs are images or video. Moreover, feeding such data to a traditional MLP poses the problem of the complete loss of information on the spatial relationship between variables.

MLPs are however quite easy to adapt to this particular type of data. Again, the source of inspiration is the animal nervous system, and in particular the primary visual cortex [79], which is organized as a series of layers, just like an MLP, but where connections between neurons are only local. This way, the fan-in of hidden units is limited to the output of a subset of units in the previous layer and spatial proximity acquires a key role in the way features are processed. *Convolutional* and *pooling* layers implement this idea in different ways.

Pooling layers perform a quite simple sub-sampling operation, usually taking the maximum, *max-pooling*, or mean, *mean-pooling*, value of each patch and using disjoint patches over the data [80].

Convolutional layers, on the other hand, use a form of discrete convolution, which

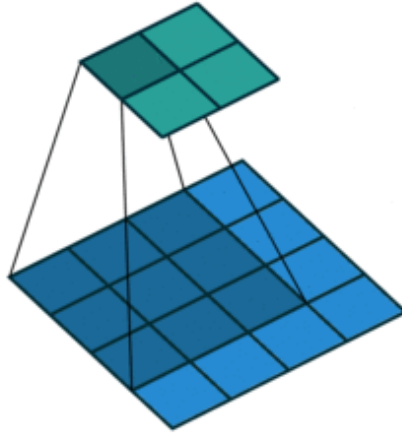


Figure 2.3.4: Discrete convolution (https://github.com/vdumoulin/conv_arithmetic). The image is represented as a blue grid being processed by a kernel represented as a dark blue grid, resulting in an output represented as a green grid.

is defined as

$$(f * g)(t) = \sum_s f(s)g(t - s),$$

for one-dimensional data, or as

$$(F * G)(t, t') = \sum_s \sum_{s'} F(s, s')G(t - s, t' - s'),$$

for two dimensional data, in place of the standard matrix product between the weights and the inputs of a hidden layer, with convolution *kernels*, also called *filters*, taking the place of weights. Patches can be disjoint or partially overlapping, and there are several different strategies for the margins of the grid defined by the data, resulting in many different types of convolutional layer being available.

The number of kernels applied is arbitrary just like the number of units in a regular, also called *dense*, layer. Figure 2.3.4 illustrates one of the different possible ways a discrete convolution operates on an image.

Two-dimensional filters are the most common as a consequence of the popularity of image-processing tasks, but depending on the data, one-dimensional, three-dimensional, or even higher-dimensional filters might be more appropriate.

A two-dimensional convolutional layer of L $K \times K$ filters will produce L representations $Z_l = f(W_l * X)$, where f is the activation function, $*$ the convolution operator, X the image and W_l the set of weights corresponding to the filter $l = 1 \dots L$. If X is an $M_1 \times M_2$ image, this will result in a $(M_1 - K + 1) \times (M_2 - K + 1)$ representation for each of the filters. Filters conceptually correspond to hidden units in a dense layer, and its number is also a free hyper-parameter. The number of weights in such convolutional layer is $L \times K^2$, while the output dimension is $L \times (M_1 - K + 1) \times (M_2 - K + 1)$, which can not only exceed that of a dense layer with a fraction of its weights, but also preserves the grid structure of the data.

If a two-dimensional max or mean pooling layer of $P \times P$ patches is applied immediately after the convolutional layer, the final output has a $L \times (M_1 - K - P +$

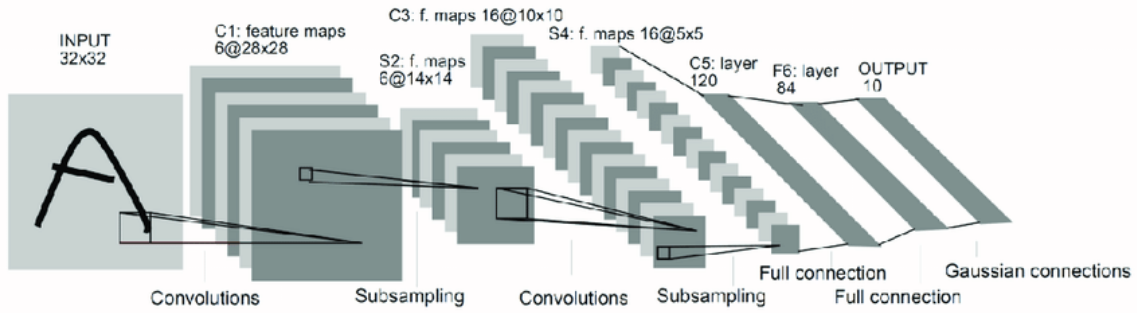


Figure 2.3.5: LeNet-5 for image processing, from [81]

$2) \times (M_2 - K - P + 2)$ dimension. In this case the operation applied is fixed and no weights need to be learnt, and the obtained effect is translation invariance and reduced overfitting.

This combined convolution–pooling process allows processing the input using a moderate number of weights while preventing overfitting and the loss of useful spatial information.

Notice that, to be effective, convolutional and pooling layers must act on inputs that have a spatial structure and are naturally distributed in feature channels. This is the case for images or video and their decomposition in RGB channels, but also of weather prediction data where the different meteorological features (pressure, temperature, wind components, etc.) correspond to different input channels.

An MLP that makes use of convolutional layers is generally called a *convolutional neural network* (CNN). Figure 2.3.5 represents *LeNet-5*, perhaps the most widely known CNN, that was developed in the 90s of the 20th century to classify handwritten digits. It is common to insert *pooling* layers between convolutional layers, with the goal of reducing the dimension of the internal representations of the data.

2.3.1.4 Temporal data and sequential patterns: The Recurrent Neural Network

Another source of information that is neglected by traditional models is temporal information represented as an order between rows of X . Time series like the value of companies in the stock exchange are usually stored as a table with columns representing the companies and rows representing timestamps. There is no internal structure of datum like in a grid representing an image, but the order of the rows is of critical importance. The traditional approach involves feature engineering with tools like *auto-regressive models*.

CNNs with one-dimensional kernels can be applied to make use of this temporal structure without the need of feature engineering, but this presents an important drawback. If the model needs a long term memory to yield competitive results, the length of the convolutional kernels might be too big to be practical, or, alternatively, a large number of convolutional layers with smaller kernels would be required, which is also a problem. In other words, the persistence of the model memory is related to the number of parameters of the model, be it through the length of the one-

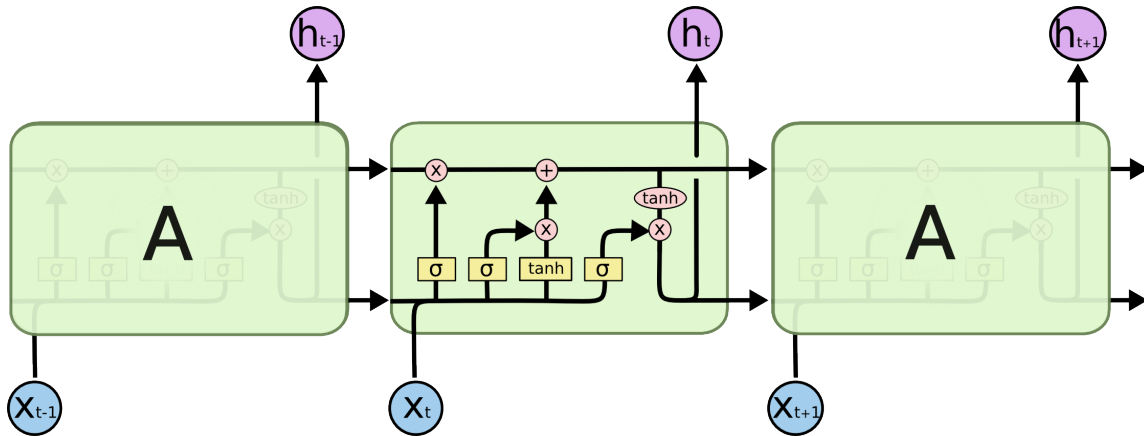


Figure 2.3.6: LSTM unit, from <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

dimensional kernels or through the number of convolutional layers applied.

In fact, this kind of data is better exploited by an artificial neural network with recurrent connections in its hidden layers. Such connections are loops that feed each unit with its own previous output, and allow the unit to have a kind of memory over the previously processed data.

The most frequent type of recurrent unit is *Long-Short Term Memory* (LSTM) that can be seen in figure 2.3.6. These units have several internal logic gates, each of them with its own parameters, so the total number of parameters in the network is much higher than in non-recurrent networks. Training recurrent networks is generally associated to a much higher computational cost than training other types of neural networks, but even so, they are considered the most powerful models for processing data with temporal structure, and are widely used in practice for natural language, voice or video processing [82].

2.3.2 Optimization

The Universal Approximation Theorem combined with results showing that depth increases expressive power [75] could lead to the idea that any problem would be easy to solve with an MLP of the correct architecture. However, in practice an MLP with a high number of hidden layers, typically more than just 3 or 4, will present a series of problems. To begin with, the training algorithm is more sophisticated than that of a linear model, since it has to deal with the layers of parameters and their sequential application to the data. This results in a more difficult hyper-parameterization of the algorithm, and the danger of underfit that increases with the number of hidden layers if not enough training iterations are performed. Also, the great expressive power of MLPs can easily lead to overfit, and several measures need to be taken to avoid it.

2.3.2.1 Back-propagation

The MLP is usually trained with the algorithm known as *back-propagation* [44, 71], which is used in conjunction with *forward-propagation*, the process of inference of the MLP, during training. For each iteration of the training algorithm, forward-propagation is used to produce predictions for a set of training data, which can then be used to compute a cost with respect to the expected targets. Back-propagation makes the information flow back through the network, layer by layer from the output to the input, to allow the computation of the gradient used to update the parameters.

In more detail, given a batch of training patterns and targets, the network will compute all the activations from the input layer to the output layer in the forward pass. Once the output values are available, an error can be calculated with the selected loss function, but that error can only be used to compute the gradient and the new weight values for the output layer units. Backpropagation defines a kind of generalized error associated to a hidden unit as the weighted average of the generalized errors of the units in the following layer. Let i, j, k be the indices of units in three consecutive layers, W_{ji} the weight of the connection from unit i to unit j , Z_i the output of unit i , $S_j = \sum_i W_{ji} Z_i$ the sum of the weighted inputs of unit j , $\delta_k = Y_k - \hat{Y}_k$ the generalized error at unit k , and F the activation function. Then

$$\frac{\partial E_L}{\partial W_{ji}} = \frac{\partial E_L}{\partial S_j} \frac{\partial S_j}{\partial W_{ji}} = \delta_j \frac{\partial S_j}{\partial W_{ji}},$$

where

$$\begin{aligned} \delta_j &= \frac{\partial E_L}{\partial S_j} = \sum_k \frac{\partial E_L}{\partial S_k} \frac{\partial S_k}{\partial S_j} \\ &= \sum_k \frac{\partial E_L}{\partial S_k} \frac{\partial S_k}{\partial Z_j} \frac{\partial Z_j}{\partial S_j} \\ &= \left(\sum_k \frac{\partial E_L}{\partial S_k} W_{kj} \right) F'(S_j) \end{aligned}$$

and in a regression model

$$\frac{\partial S_j}{\partial W_{ji}} = Z_i,$$

so for the output layer, where the error can be computed directly,

$$\frac{\partial E_L}{\partial W_{ji}} = \delta_j Z_i,$$

and for the hidden layers

$$\frac{\partial E_L}{\partial W_{ji}} = \left(\sum_k \frac{\partial E_L}{\partial S_k} W_{kj} \right) F'(S_j) Z_i = \left(\sum_k \delta_k W_{kj} \right) F'(S_j) Z_i.$$

So, with backpropagation the error of the output units is computed first, then the general error of the units of the last hidden layer, and successively all the previous hidden layers. This is the *backward pass* of the learning algorithm.

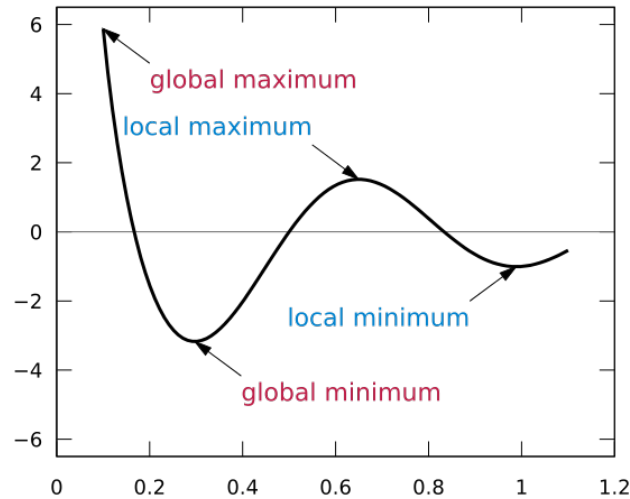


Figure 2.3.7: Local and global optima in a loss function, from Wikimedia Commons (distributed under a CC BY-SA 3.0 license)

The computational cost of backpropagation for a multilayer perceptron with L layers, all of them with M units, and a size N training dataset of dimension d is

$$\mathcal{O}((L - 1) \times M^2 \times N) + \mathcal{O}(d \times M \times N).$$

Notice that the derivative of the activation function is needed in the calculations, so the activation function must be differentiable, or at least differentiable almost everywhere. As was noted earlier, the activation function of the output layer is usually determined by the problem at hand, being the identity preferred for regression problems and a logistic (or softmax) the usual choice for binary (or multiclass) classification, while the activation of the hidden layers is a free hyper-parameter. However, not every differentiable function is suitable for the task. In particular, linear functions shouldn't be used as activations of the hidden layers, since the composition of linear functions is also a linear function, and a MLP with linear activations in all its hidden layers is equivalent to a single layer perceptron in terms of expressive power, as was discussed before, but with the additional disadvantage of having a higher complexity.

2.3.2.2 Underfit

In the case of linear models and the usual loss functions, the function to minimize on the model parameters is convex. This means that a unique global minimum exists and it is relatively easy to find. However, in an MLP, which consists in a composition of non-linear functions, this convexity is lost, and local minima are present in the non-convex objective function. This will lead to a number of difficulties when trying to reach the optimal solution, including getting stuck in sub-optimal local minima like those represented in figure 2.3.7. A series of particular problems related to underfit and proposed solutions are outlined in what follows.

Local minima and advanced optimization algorithms

Many advanced optimization algorithms have been developed in order to mitigate the difficulties derived from the existence of local minima in the loss function.

A way to accelerate convergence is through the use of second order derivatives in addition to the first order derivative used by SGD. However, the computational cost of the Hessian is too high and usually some simplifications are used instead [71, 83].

Others use *momentum* to escape local minima without incurring in too high costs, or Nesterov's Accelerated Gradient as an improvement on the classical momentum enhancement [50].

Finally, other methods [84, 51, 52, 53] dynamically adjust the learning rate during training, trying to escape from local minima by increasing it, and to approximate the global minima by reducing it near the end of the training. Also, getting rid of the learning rates used in SGD and that are often difficult to adjust in order to achieve effective training eases the costly process of hyper-parameter search. Most of these techniques can be traced back to work by J. LeCun *et. al.* [71].

Several variants of this ideas have been developed such as Adaptive Subgradient (AdaGrad) [51], AdaDelta [52] or Adaptive Moment Estimation (Adam) [53] that can be seen as an improvement over AdaGrad and AdaDelta and is maybe the most popular optimization method for ANNs today, as it combines the use of momentum with a dynamically adjusted learning rate.

Vanishing gradient, weight initialization and activation functions

As noted before, a mandatory condition for an MLP to be more expressive than a linear model is the non-linearity of the activation functions used in the hidden layers. It is usual, then, to employ logistic or, even better due to its symmetry with respect to the origin, hyperbolic tangent as activation functions.

However, such functions are quite costly in computational terms and, more importantly, the gradient of such functions takes values in $[0, 1]$, and saturates very fast when moving away from the origin. So, if the activation function of each unit is not kept in the “almost linear” region (the “middle” region of the sigmoid or hyperbolic tangent) it is easy to make it saturate and shrink the gradient, slowing training. Also, this fact results in progressively smaller gradients as the chain rule is applied to adjust the weights of layers from the output to the input. As a consequence, layers far from the output are trained very slowly.

On the other hand, a poor initialization of the parameters could get the training algorithm stuck in a local minima or a plateau of the loss function. Quite often, the random initialization with the traditional probability distributions [71] will lead to such situations. The combination of a bad choice as activation function and a poor initialization of the parameters results in the phenomenon known as the *Vanishing Gradient* [34].

To alleviate the first problem, the use of functions that do not saturate like *softsign* or *Exponential Linear Unit* (ELU), or even rectified functions like *Rectified Linear Unit* (ReLU), is much more popular nowadays. Also, such functions are cheaper to compute than traditional sigmoid functions, which is another important advantage.

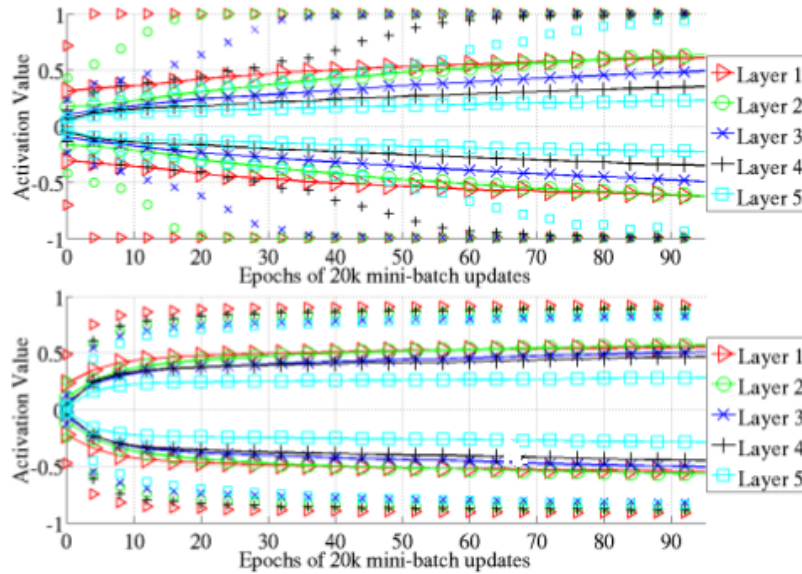


Figure 2.3.8: Layer activations during training [34], percentiles 98 (markers) and standard deviation (lines with markers) of the activations in different layers of a MLP for *tanh* (up) or *softsign* (bottom) activation function.

Figures 2.3.8 and 2.3.9 illustrate the different behaviors of an MLP layers depending on the activation function used. Using a function that saturates slowly (*softsign*), such activations take more varied values away from 0.

On the other hand, it is also necessary to mitigate the problem related to the random initialization of the weights for the training to succeed. Traditionally [71] a normal distribution with mean 0 and standard deviation of a factor of $\frac{1}{\sqrt{m}}$ with m the number of units of the previous layer was used to keep the activations in their linear regime during the first iterations of training.

It was shown in [34] that bad initializations resulted in gradients centered at 0 and whose variance decreased as one moved from the output to the input layers. Thus, the network was stuck at weights that caused this near zero gradient behavior in the first layers and that were therefore unable of any further learning.

In [34], Glorot and Bengio proposed a layer initialization taking into account the number of units in both the previous and following layer and the activation function that mitigated the vanishing gradient problem by keeping the variances of backpropagated gradients stable. In particular, if M_i and M_{i+1} are the fan-in and fan-out of the i -th layer that has hyperbolic tangent activation, the uniform distribution suggested for initialization would be

$$U \left[-\frac{\sqrt{6}}{\sqrt{M_i + M_{i+1}}}, \frac{\sqrt{6}}{\sqrt{M_i + M_{i+1}}} \right]. \quad (2.3.1)$$

Figures 2.3.10 and 2.3.11 show how, if the initialization is not optimal, gradients concentrate in values closer to 0 in the layers far from the output, while the activations have the opposite behavior, being smaller near the output. Both phenomena make training much more difficult.

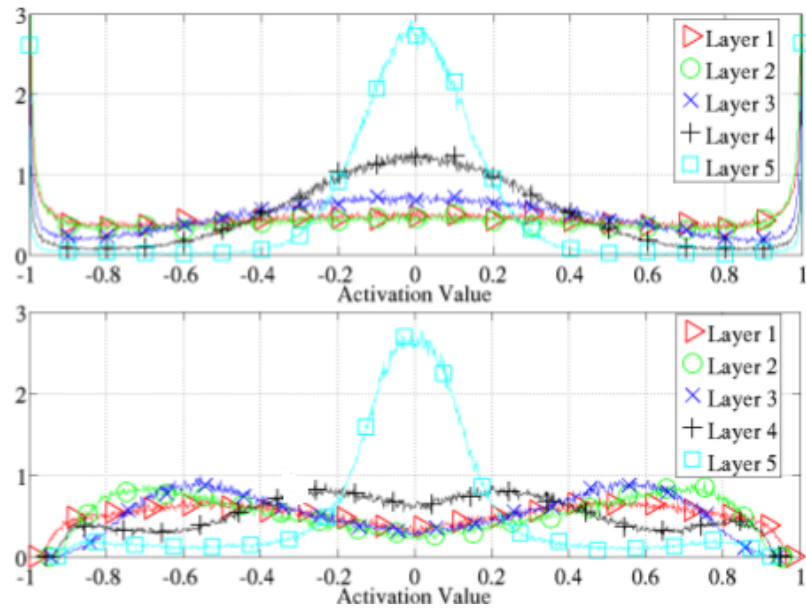


Figure 2.3.9: Layer activations after training [34], histograms of activations of different layers in a trained MLP with *tanh* (up) or *softsign* (bottom) activation function.

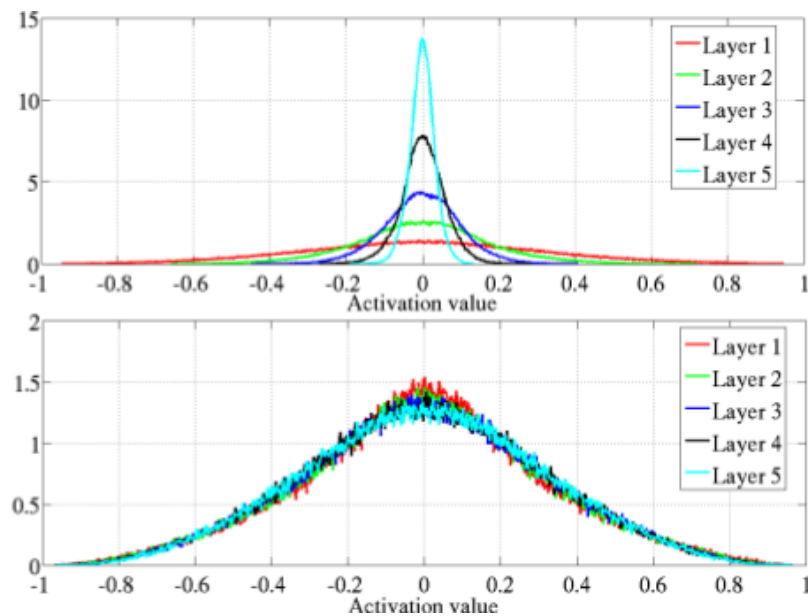


Figure 2.3.10: Distribution of activations [34], distribution of the activations for each layer with classical initialization (up) and Xavier initialization (bottom).

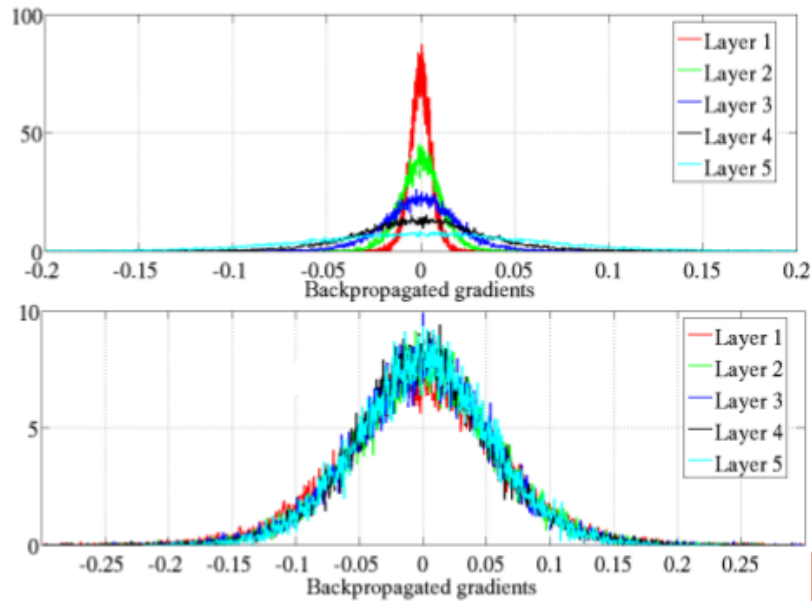


Figure 2.3.11: Distribution of gradients [34], distribution of gradients for each layer with classical initialization (up) and Xavier initialization (bottom).

Internal Covariate Shift and Batch Normalization

On the other hand, when a linear model is trained with gradient descent, it is of paramount importance to previously prepare the data by rescaling each feature so that all of them have the same magnitudes. This is usually achieved by standardizing the data, and the model can, then, be safely trained so it will be able to produce useful predictions over new, unseen, standardized data.

Training an MLP in such fashion with backpropagation and gradient descent, the first hidden layer will receive standardized data, but each of the following layers receives the activations of the previous layer, which are not standardized. The effect this has over the model is similar to that observed in a linear model when the input data is not rescaled, and makes training considerably harder. Also, the probability distribution of the activations of a layer needs not be normal, and will change as long as the weights are updated in each training iteration, aggravating the situation. This problem is known as *Internal Covariate Shift* [54].

A solution to this problem, the technique known as *batch normalization*, has been proposed. It consists in standardizing the activations of each hidden layers for each batch during training.

2.3.2.3 Overfit

Another recurrent problem with MLPs is how prone they are to overfit, due to their ability as universal approximators. An overfit model will yield a very low training error, but will not generalize well and its test or cross-validation error will be poor. Different sub-problems and proposed solutions are detailed in what follows.

Excessively long training and Early Stopping

If the model is complex enough, i.e., has enough layers of sufficient number of units, training it for a too large number of epochs will usually result in overfit. During training, a partition of the data can be saved so it is not used to adjust parameters but only to measure an error metric. This *validation* partition is used to estimate the generalization error of the model. If an increase in the validation error is observed, then the MLP is overfitting and training can be safely ended [19]. This stopping mechanism is known as *Early Stopping*, and is extremely popular, to the point of being included in most Machine Learning software libraries like Scikit-learn [20] and Keras [66].

Excessive number of parameters and regularization

A model that is too complex due to an excessively large number of parameters will be prone to overfit, but knowing in advance the optimal number of parameters and architecture for a particular problem is not possible. On the other hand, performing hyper-parameter search over all the hyper-parameters related to the architecture of a model (number of hidden layers, units per layer, etc.) is a daunting task. However, a search over a single regularization hyper-parameter can be much simpler and more convenient, and affect performance of the model in a similar way. An objective function can be modified to include an additional term that keeps the total sum of the parameters controlled. This term can be of many forms, but it is usually one of the following:

\mathcal{L}^1 (*Lasso*): of the form $\lambda \sum_{ij} |w_{ij}|$.

\mathcal{L}^2 (*Ridge, Tikhonov, weight decay*): of the form $\lambda \sum_{ij} (w_{ij})^2$.

where λ is the coefficient that adjusts the intensity of regularization. As an example, a regression problem where squared error and \mathcal{L}^2 regularization are used corresponds to the following objective function:

$$E = \frac{1}{N} \sum_{n=1}^N (\mathbf{y}_n - \hat{\mathbf{y}}_n)^2 + \lambda \sum_{ij} (w_{ij})^2. \quad (2.3.2)$$

When performed only in the last layer and with linear outputs, this is equivalent to the ridge regression fit of the deep features generated by the hidden layers of the network.

Also, other more sophisticated regularization techniques have been developed. Adding gaussian noise to the activations of the hidden layers [85] or randomly deactivating selected units inside the MLP during training usually result in models more robust and resilient to overfitting. The technique known as *dropout* [48] is specially effective to avoid overfit. It consists in, for each iteration of the training algorithm, and both in forward propagation and backward propagation, deactivating each unit with probability $1 - p$. The effect this has over the MLP is similar to that of making an ensemble of MLPs, but with a fraction of the cost. Once trained and when

it comes to inference, the model will not deactivate units, but all weights will be multiplied by p so the model will remain deterministic while maintaining the mean values of activation it had during training.

Chapter 3

Deep Neural Network Prediction in Wind and Solar Energy

3.1 Summary

As mentioned, Deep Neural Networks possess very powerful modeling abilities, due to the fact that they are universal approximators even with a single hidden layer. These abilities become even more impressive when an input having a multidimensional grid-like structure is captured and exploited by convolutional layers.

This first series of publications explore the application of Artificial Neural Networks to the prediction of wind energy production and daily solar radiation with inputs obtained from Numerical Weather Prediction (NWP) [25] systems, that present the two-dimensional grid structure suitable for Convolutional Neural Networks (CNNs). At the time, they were among the first contributions to make use of the new DNN frameworks for renewable energy prediction.

Both classical Multilayer Perceptrons (MLPs) and CNNs are considered in combination with modern techniques like proper weight initializations, dropout regularization or weight decay, and improved with the use of ensembles. As will be proved, such models can improve on Support Vector Regression, which can be considered the state of the art for this type of regression problems. Moreover, while the datasets considered are relatively small and thus the computational costs of training SVRs are similar to those of training MLPs and CNNs, it is expected that newer and bigger datasets will be available with time, and thus neural networks will become a more interesting option than SVRs due to the fact that their training costs grow linearly with the size of the training dataset.

3.2 Publications

We detail next the publications of this chapter, give their abstracts, and briefly list their main contributions for the sake of convenience.

Main publication

- David Díaz-Vico, Alberto Torres-Barrán, Adil Omari, and José R. Dorronsoro. Deep neural networks for wind and solar energy prediction. *Neural Processing Letters*, page 1–16, 04 2017

Abstract: Deep Learning models are recently receiving a large attention because of their very powerful modeling abilities, particularly on inputs that have a intrinsic one- or two-dimensional structure that can be captured and exploited by convolutional layers. In this work we will apply Deep Neural Networks (DNNs) in two problems, wind energy and daily solar radiation prediction, whose inputs, derived from Numerical Weather Prediction systems, have a clear spatial structure. As we shall see, the predictions of single deep models and, more so, of DNN ensembles can improve on those of Support Vector Regression, a Machine Learning method that can be considered the state of the art for regression.

Contributions: This article was published as an extension of [27]. Going deeper into the study of the performance of CNNs in regression for renewable energy-related problems, the main contributions were:

- The use of ensemble techniques to futher improve the single CNN results. The significant improvements obtained indicate that techniques such as bagging can also be useful when working with strong learners such as deep neural networks, possibly due to the randomness induced in these models by their initialization, minibatch training and dropout regularization.
- The use of very recent at the time optimization methods such as Adadelta and other techniques like rectified activations and dropout regularization, that can substantially improve the results obtained.
- The application to solar radiance prediction in addition to prediction of wind energy production, with data from Oklahoma’s Mesonet network available at [86].
- The use of more modern and solid technologies as Keras [66], combined with Theano [60] and Tensorflow [58], in substitution of the older and less flexible Pylearn2 [65] used in [27].

Other publications

- David Díaz, Alberto Torres, and José R. Dorronsoro. Deep neural networks for wind energy prediction. In *Advances in Computational Intelligence - 13th International Work-Conference on Artificial Neural Networks, IWANN 2015, Palma de Mallorca, Spain, June 10-12, 2015. Proceedings, Part I*, pages 430–443, 2015

Abstract: In this work we will apply some of the Deep Learning models that are currently obtaining state of the art results in several machine learning problems to the prediction of wind energy production. In particular, we will consider both deep, fully connected multilayer perceptrons with appropriate weight initialization, and also convolutional neural networks that can take advantage of the spatial and feature structure of the numerical weather prediction patterns.

We will also explore the effects of regularization techniques such as dropout or weight decay and consider how to select the final predictive deep models after analyzing their training evolution.

Contributions: This congress paper detailed the arguably first case of application of CNNs to the prediction of wind energy production. Different neural networks with MLP and CNN architectures were compared against SVRs, which could be considered the state of the art, for the problem of prediction of wind energy production with data provided by ECMWF for both the Sotavento wind farm and peninsular Spain. Results proved that CNNs can exploit the grid-like structure of NWP data and improve on the results of other classical models, SVRs in particular, while MLPs, although being universal approximators, fall below even when their number of parameters is higher. While higher computational cost of CNNs compared to SVRs for this dataset, being relatively small, seems an important disadvantage of the proposed solution, the situation will be the opposite when working with big enough datasets due to the fact that the cost of the training for neural networks grows linearly with the size of the training dataset. Pylearn2 [65] was extensively used in this paper experiments, but would be discarded in favor of more flexible high level libraries in the following publications.

Deep Neural Networks for Wind and Solar Energy Prediction

David Díaz–Vico, Alberto Torres–Barrán, Adil Omari & José R. Dorronsoro

Neural Processing Letters

ISSN 1370-4621

Volume 46

Number 3

Neural Process Lett (2017) 46:829-844

DOI 10.1007/s11063-017-9613-7



Your article is protected by copyright and all rights are held exclusively by Springer Science +Business Media New York. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your article, please use the accepted manuscript version for posting on your own website. You may further deposit the accepted manuscript version in any repository, provided it is only made publicly available 12 months after official publication or later and provided acknowledgement is given to the original source of publication and a link is inserted to the published article on Springer's website. The link must be accompanied by the following text: "The final publication is available at link.springer.com".



Deep Neural Networks for Wind and Solar Energy Prediction

David Díaz–Vico¹ · Alberto Torres–Barrán¹ ·
Adil Omari¹ · José R. Dorronsoro¹

Published online: 1 April 2017
© Springer Science+Business Media New York 2017

Abstract Deep Learning models are recently receiving a large attention because of their very powerful modeling abilities, particularly on inputs that have a intrinsic one- or two-dimensional structure that can be captured and exploited by convolutional layers. In this work we will apply Deep Neural Networks (DNNs) in two problems, wind energy and daily solar radiation prediction, whose inputs, derived from Numerical Weather Prediction systems, have a clear spatial structure. As we shall see, the predictions of single deep models and, more so, of DNN ensembles can improve on those of Support Vector Regression, a Machine Learning method that can be considered the state of the art for regression.

Keywords Deep learning · Convolutional neural network · Wind energy · Solar energy

1 Introduction

Artificial Neural Networks (ANNs) have had three moments in the limelight. The first one in the second half of the 50's, full of a somewhat naïve promise, was followed by a long winter from the mid 60's to the mid 80's. The second started around 1990, where ANNs received a large interest that led to a very good understanding of neural networks with one or two layers and that established ANNs as the state of the art approach for classification and regression problems. However, the appearance of new and competing modeling proposals

David Díaz–Vico
david.diazv@estudiante.uam.es

Alberto Torres–Barrán
alberto.torres@uam.es

Adil Omari
adil.omari@inv.uam.es

José R. Dorronsoro
jose.dorronsoro@uam.es

¹ Departamento de Ingeniería Informática e Instituto de Ingeniería del Conocimiento, Universidad Autónoma de Madrid, Madrid, Spain

(Support Vector Machines, Boosting, Random Forests) and the inability to train efficiently MLPs with three or more layers because of vanishing gradients, led to a mild decline in the research and applications of ANNs around the year 2000.

The third limelight moment for ANNs has come with the by now very famous Deep Neural Networks (DNNs) [22]. While they can be defined as ANNs with several (at least three or more) hidden layers and, thus, seen as just a simple enlargement of previously studied architectures, their success has been enormous, as they have shattered records in image and speech recognition competitions, make the core of the famous Deep Mind system for playing Go and are currently held among the most promising building blocks towards the still elusive goal of achieving Artificial Intelligence.

The first crucial step in this third epoch was the proposals by Hinton and Salakhutdinov [17] on the one hand, and by Bengio et al. [5] on the other, to overcome the vanishing gradient obstacle and to make possible the effective training of DNNs for the first time. Subsequent work has given DNNs a tremendous impulse in which a breakthrough has led to another. First, better initialization procedures such as Glorot's [13] greatly simplified the somewhat clumsier pretrainings in [17] and [5] and made possible to train large networks by backpropagation. In turn, new regularization procedures, particularly dropout [29] allowed to control the clear risk of overfitting present in the very large networks that were now possible. New input processing schemes were next introduced, such as the rediscovered convolutional layers, first proposed by Y. LeCun in the late 1990's, or the Rectified Linear Unit (ReLU) activations [14], whose 0–1 derivatives lend stability to training. Training itself has also greatly changed and improved. The sensible (but modest) minibatch training has been complemented by new and powerful gradient descent techniques, such as Nesterov's variant of classical momentum [30], self adjusting learning rate procedures such as Adagrad [11], Adadelata [31] or Adam [20], or batch normalization techniques to control covariance shift [2, 18].

In parallel, key advances have been made in two crucial areas. The first one is the introduction and extensive use of GPU libraries to speed up the very costly training of large networks by vectorializing the huge number of matrix-vector operations needed; in turn, this has been coupled with new proposals and implementations for distributed DNN training [1, 25]. The second advance, extremely important but sometimes missed, is the development of symbolic differentiation compilers that can automatically compute and yield very efficient low level code for the highly complex gradients associated to very general feedforward architectures, specially suited to concrete problems which are now routinely proposed. These two advances have resulted in a growing number of publicly available training software platforms, either at a relative low level, such as Caffe [19], Pylearn2–Theano [3, 6] or Google's TensorFlow [1], or as high level efficient wrappers such as Keras [9] that runs on top of either Theano or TensorFlow backends (we will use both Pylearn2 and Keras for our experiments here).

The main consequence of all this has been an ongoing, tremendous research effort, with very impressive results in a number of areas, particularly on problems from computer vision and speech recognition. The extensive use of convolutional layers is a key factor here, but a deeper reason may be the formal similarity between the information processing of ad-hoc DNN architectures and the layer processing that takes place in the visual cortex [21]. In the DNN case, this processing results in successively refined representations of input patterns that at the last hidden layer are powerful enough to be successfully exploited by simple readouts. In other words, DNN training can be seen as a particularly effective way to perform feature engineering, to the point that the field is often identified with representation learning [4].

As just mentioned, DNNs are most often used in image or speech recognition problems while less attention has been comparatively paid to other problems whose inputs also have a bidimensional, image-like structure. An example is given by problems with Numerical

Weather Prediction (NWP) patterns. NWP provides forecasts for a geographical area as a number of weather variable predictions given at each one of the points of a rectangular grid. Each such variable can be thus seen as a particular kind of image of the area under consideration or, in convolutional network language, as a feature map over a concrete channel. This image-like structure naturally suggests that DNN architectures similar to those used in image processing may result in good regression models. We will consider here two kind of such problems. The first is forecasting wind energy production, whose increasing penetration in many countries gives a great importance to its accurate prediction. This is an intensely studied problem where standard MLPs and Support Vector Regression (SVR) acting on NWP forecasts provided by organizations such as the European Center for Medium range Weather Forecasts (ECMWF, [12]) or the Global Forecasting System (GFS, [27]), are the models most often applied. Here we will consider wind energy predictions at the farm level (namely, the Sotavento farm in Northwestern Spain) and on a wide area (namely wind energy production of peninsular Spain). The NWP inputs here will be those provided by the ECMWF.

Our second problem will be the prediction of total daily incoming solar radiation where we will use data from the recent Kaggle *AMS 2013–2014 Solar Energy Prediction Contest* [28], whose goal was to predict aggregated incoming radiation on a total of 98 Mesonet weather stations covering the state of Oklahoma using as inputs NWP forecasts provided by NOAA/ESRL Global Ensemble Forecast System (GEFS).

It is clear that for both problems convolutional networks arise as natural choices to derive energy forecasts and they will be used in the deep models considered in this paper. It is also well known that a good hyper-parameter selection is crucial when applying any Machine Learning (ML) model, and this is the case too of DNNs, with the extra difficulty of the potentially very large number of hyper-parameters as well as the large number of processing options that have been proposed in the literature. To simplify on this we will work in a relatively standard setting, using Glorot–Bengio weight initialization [13], ReLUs [14], dropout regularization [29] on hidden layers and standard weight decay in the final ones. As mentioned, several training algorithms can be used; we will settle with Adadelta [31], which allows for an essentially self-adjusting learning rate. The exploration of the other relevant hyper-parameters is handled with Hyperopt [7], a recent tool that allows for a principled random exploration of the hyper-parameter space. We point out that this contribution is a substantially larger extension of previous work [10] by us. Our contributions here thus further extend and enlarge those in [10] and can be summarized as follows:

- We extend and update the review in [10] of the most recent proposals in DNNs, in particular those for DNN training. While known, the techniques we review are scattered among many different papers and our joint presentation of them will be helpful for readers that are considering to use DNNs.
- We build on the techniques reviewed to set up a simple and useful methodology for DNN training and hyper-parameter selection in regression problems.
- We will thoroughly explore the application of convolutional DNNs to the wind energy and solar radiation problems from the point of view of the ML practitioner.
- We introduce DNN ensembles as a way to enhance single DNN predictions by lowering variance while retaining a good enough bias and show experimentally how the random elements of DNN training (random weight initialization, minibatch training and dropout regularization) result in robust and effective DNN ensembles.

As mentioned before, we will use the Pylearn2 [15]–Theano [3,6] and Keras–Theano [9] platforms as they include a wide variety of already tested neural networks and allows us to explore several of the latest and most effective proposals for deep network training. Besides

its symbolic differentiation capabilities, that make possible the automatic computation of the cost function gradients of fairly complex networks, another key advantage of having Theano as the underlying backend is that we can exploit the final low level GPU code it generates to greatly speed up DNN training with respect what is possible over standard CPUs.

The rest of the paper is organized as follows. In Sect. 2 we review our choices for deep network configuration and optimization procedures and discuss some of their details. Section 3 contains a succinct discussion of the framework for wind energy prediction over NWP inputs, a description of our experimental setup and the prediction results for both the Sotavento wind farm and the entire wind energy prediction over peninsular Spain that is overseen by Red Eléctrica de España (REE). For the Sotavento problem we improve on the results of our previous work [10] by working with the much more flexible Keras DNN wrapper, simplifying hyper-parameter selection and introducing DNN ensembles that further improve our previous single DNN prediction errors. We build on the general approach for wind energy to deal in Sect. 4 with the solar radiation problem proposed in the Kaggle *AMS 2013–2014 Solar Energy Prediction Contest* [28], showing that single DNN and ensemble models can also improve on SVRs on this setting. Finally, in 5 we briefly discuss our results and offer pointers to further work.

2 Deep Neural Networks

We briefly review here some of the key issues when configuring and training Deep Neural Networks, closely following in the first four subsections our earlier presentation in [10]; Sect. 2.5 is new.

2.1 Initialization

It can be said that the problem of vanishing gradients that plagued the backpropagation training of networks with more than 2 layers was in a great measure caused by bad weight initialization. In fact the breakthroughs of Hinton and Bengio mentioned above were ultimately clever ways to initialize a DNN in such a way that subsequent backpropagation was successful. This was followed by the work in [13] where it was shown that bad initializations resulted in gradients centered at 0 and whose variance decreased as one moved from the output to the input layers. Thus, the network was stuck at weights that caused this near zero gradient behavior in the first layers and that were therefore unable of any further learning.

Glorot and Bengio proposed in [13] a simpler way to initialize DNN weights so that vanishing gradients are avoided. Their starting point is LeCun's work in [24], where it is suggested to use a properly normalized hyperbolic tangent activations and to draw the initial weights from a uniform distribution $U\left[-\frac{\sqrt{3}}{\sqrt{M}}, \frac{\sqrt{3}}{\sqrt{M}}\right]$ so that the (linear) activations and (non linear) outputs of a neuron are kept in the $[-1, 1]$ active range of the (normalized) hyperbolic tangent. Following on this, it is shown in [13] that, provided the initial weights W_i verify

$$M_i \text{Var}(W_i) = 1; \quad M_{i+1} \text{Var}(W_i) = 1 \quad (1)$$

where M_i and M_{i+1} are the fan-in and fan-out of the units in the i -th layer, one can achieve that $\text{Var}(z_i) \simeq \text{Var}(z_j)$ across the successive z_j layers and also that $\text{Var}\left(\frac{\partial J}{\partial z_i}\right) \simeq \text{Var}\left(\frac{\partial J}{\partial z_k}\right)$ for the preceding z_k layers, where J denotes the MLP cost function. In particular, the variances of backpropagated gradients remain stable and vanishing gradients will no longer appear.

Obviously, a reasonable trade-off between both terms in (1) is to take $\text{Var}(W_i) = \frac{2}{M_i + M_{i+1}}$, i.e., to initialize the W_i using an uniform distribution

$$U \left[-\frac{\sqrt{6}}{\sqrt{M_i + M_{i+1}}}, \frac{\sqrt{6}}{\sqrt{M_i + M_{i+1}}} \right], \tag{2}$$

which coincides with the above mentioned initialization proposed in [24] when $M_i = M_{i+1}$. Ultimately all this is related to the Batch Normalization proposals in [18] and Propagation Normalization in [2] to correct covariance shift.

Here we will use Rectified Linear Unit (ReLU) activations, discussed next, instead of the hyperbolic tangent ones but, nevertheless, will also apply the Glorot–Bengio initialization using the suggestion in [16] to dilate the Glorot–Bengio uniform intervals by a factor of 1.5. In fact, we have observed that in accordance with the analysis in [16], this usually yields better results.

2.2 Activation Function

We have used linear units in the output layer and ReLU units in the hidden layers. This decision is motivated by the fact that ReLU units don't face vanishing gradient problem in the same way as units with different activation functions like sigmoid or tanh do. The ReLU activation is defined as $r(x) = \max(0, x)$. Briefly speaking, it is a piecewise linear function which switches to zero negative inputs and preserve the positive ones. The ReLU activation brings several advantages to our models: accelerated convergence of gradient descent methods, a notable help to avoid the vanishing gradient problem and induced sparsity in the representations of the successive layers. On the other hand, ReLUs share some similarities with functions relating neuronal input currents and firing rates that appear in the leaky integrate and fire models used in biological neuron models [14].

2.3 Regularization

Adding regularization to Deep Neural Networks is often mandatory due to the extremely large number of weights. The standard regularization technique, weight decay, consists on adding the squared norm of the weights to the objective function. When performed only in the last layer and with linear outputs, this is equivalent to the ridge regression fit of the deep features generated by the hidden layers of the network. In this work we will use more modern techniques such as dropout [29], described next. Let a_i^l be the i -th activation of the l -layer and z_i^l the corresponding output, the standard feedforward processing would yield $z_i^l = f(a_i^l) = f(w_i^l z^{l-1} + b_i^l)$, where f is the activation function. However, with dropout, a 0–1 vector r^l is first generated applying a Bernoulli distribution componentwise. The feedforward process then becomes

$$z_i^l = f(a_i^l) = f \left(w_i^l (z^{l-1} \odot r^l) + b_i^l \right), \tag{3}$$

where \odot denotes the componentwise product. Each element in r^l has a probability p of being 1, so dropout can be seen as sub-sampling a larger network at each layer. The output errors are backpropagated as in standard MLPs for gradient computations and the final optimal weights w^* are downscaled as $w_f^* = pw^*$ to yield the final weights used for testing. Dropout clearly induces a regularization of the network weights. Besides, it is reminiscent to the well known bagging technique for ensembles that repeatedly subsamples data to build specific models and then takes the average. However, in dropout all the “models” (i.e., the particular

feedforward Bernoulli realizations) share weights and they are “trained” in a single step. Although we will not use it, in [29] it is also suggested that network performance improves when dropout is combined with a bound on the L_2 norm of the weights, i.e., when they are constrained as $\|w\|_2 \leq c$, with c a second tunable parameter on top of the Bernoulli probability p .

2.4 Convolutional Layers

While traditional deep MLP architectures usually benefit from the use of a high number of units in the hidden layers, this leads to a high number of weights, $M \times M'$ if an M unit layer is connected to an M' unit one, a number that can become rather large if, for instance, inputs are images or video. Also, feeding such data to a traditional MLP poses the problem of the complete loss of information on the spatial relationship between variables.

Convolutional and pooling layers arise as a way to avoid both problems by limiting the fan-in of a given hidden unit to the output of just a subset of units in the previous layer. The definition of such a restricted fan-in is, in general, problem-dependent, but when data have an intrinsic spatial structure, a natural approach to limit the connections is to work over small patches of variables that are local in space.

Let's assume inputs to be arranged in one channel with a two-dimensional $M_1 \times M_2$ structure, and consider $K \times K$ patches over this data. Patches can be either disjoint or partially overlapping, but for simplicity we will consider a $S = 1$ stride, the displacement applied when we move from one patch to another, in both dimensions. Then there are such $(M_1 - K + 1) \times (M_2 - K + 1)$ (overlapping) patches x_j .

A convolutional layer transform consists in deriving a feature patch $p_j = f(w * x_j + b)$, where f is the activation function and $*$ denotes the convolution operator between the $K \times K$ filter w with bias b and the patch x_j . This transforms an $M_1 \times M_2$ input X into an $(M_1 - K + 1) \times (M_2 - K + 1)$ output X' . It is usual to learn a number L of filter pairs (w_l, b_l) , which conceptually correspond to the hidden units in a classical densely connected layer, since its number is a free hyper-parameter that needs similar tuning. Thus, the number of weights in a convolutional layer is $L \times K^2$, while the output dimension is $L \times (M_1 - K + 1) \times (M_2 - K + 1)$, which for $L > K$ might exceed the expressive power of a densely connected layer with a fraction of its weights, and still preserve the spatial information of the data.

A second transform, known as pooling (or subsampling), is usually applied in order to gain translation invariance and reduce overfitting. In this case, an operation such as averaging or computing the max is applied on $P \times P$ patches of X' to derive the final output X_C , which has a $L \times (M_1 - K - P + 2) \times (M_2 - K - P + 2)$ dimension. Note that since the operation applied is fixed, no weights need to be learnt for this layer.

This combined convolution-pooling process allows processing the input using a moderate number of weights while preventing overfitting and the loss of useful spatial information. Notice that, to be effective, convolutional and pooling layers must act on inputs that have a spatial structure and are naturally distributed in feature channels. This is the case for images or video and their decomposition in RGB channels, but it also happens here, given the spatial structure of weather prediction and that we can see the different meteorological features (pressure, temperature, wind components, etc.) as corresponding to different input channels.

2.5 New Minimization Approaches

Deep Networks have also resulted in several new minimizing approaches being proposed and applied, several of them borrowed from recent advances in convex optimization. Some,

such as the use of Nesterov's Accelerated Gradient [30], improve on the classical momentum enhancement of stochastic gradient descent (SGD). Other procedures essentially aim to get rid of the learning rates used in SGD and that are often difficult to adjust in order to achieve effective training. Most of these techniques can be traced back to work by LeCun et al. [24] and seek to replace standard learning rates by adaptive ones, based ultimately on Newton's method for second order optimization.

In order to avoid costly Hessian computations, these methods apply several simplifications, the first one being the consideration of just the Hessian diagonal of the network's cost function e or its Gauss-Newton approximation, working with rates of the form

$$\eta_{ij} = \frac{\eta}{\frac{\partial^2 e}{\partial w_{ij}^2} + \epsilon} \simeq \frac{\eta}{\left(\frac{\partial e}{\partial w_{ij}}\right)^2 + \epsilon} \tag{4}$$

where ϵ and η can be kept fixed, and the term $\left(\frac{\partial e}{\partial w_{ij}}\right)^2$ adapts the overall rate to the geometry of the error function, being large in low curvature dimensions (i.e., when $\frac{\partial e}{\partial w_{ij}}$ is small) and small in high curvature dimensions (i.e., when $\frac{\partial e}{\partial w_{ij}}$ is large). Variants of this basic ideas have been derived from convex optimization and play an important role in DNN training, such as Adagrad [11], which considers weight updates of the form

$$w_{ij}^{t+1} = w_{ij}^t - \frac{\eta}{\sqrt{\sum_{s=1}^t (g_{ij}^s)^2}} g_{ij}^t \tag{5}$$

where $g_{ij} = \frac{\partial e}{\partial w_{ij}}$ and the denominator approximates the average $E[g_{ij}]$. Zeiler's Adadelta [31], which will be the optimizer used in our experiments, improves on this by working with updates

$$w_{ij}^{t+1} = w_{ij}^t - \eta \frac{\frac{1}{t} \sqrt{\sum_1^t (\Delta w_{ij}^s)^2 + \epsilon}}{\frac{1}{t} \sqrt{\sum_1^t (g_{ij}^s)^2 + \epsilon}} g_{ij}^t = w_{ij}^t - \eta \frac{RMS_t(\Delta w_{ij}^s)}{RMS_t(g_{ij}^s)} g_{ij}^t \tag{6}$$

that add a momentum-like term to the numerator and where avoiding storing momentum/gradient info is avoided updating $RMS_t(g_{ij}^s)^2$ and $RMS_t(\Delta w_{ij}^s)^2$ as

$$RMS_t(g_{ij}^s)^2 = (1 - \rho) \left[RMS_{t-1}(g_{ij}^s) \right]^2 + \rho (g_{ij}^t)^2; \tag{7}$$

$$RMS_t(\Delta w_{ij}^s)^2 = (1 - \rho) \left[RMS_{t-1}(\Delta w_{ij}^s) \right]^2 + \rho (\Delta w_{ij}^t)^2 \tag{8}$$

for an appropriate ρ . As mentioned in [31], ϵ and ρ can be used with fixed values (10^{-6} and 0.95, respectively, are recommended in [31]) while η is more problem dependent.

3 Wind Energy Experiments

In this section we will apply DNNs to the problem of predicting wind energy production, first on the Sotavento wind farm and then over peninsular Spain

3.1 Wind Energy Data

We will work with the following eight NWP variable forecasts given by the European Centre for Medium-Range Weather Forecasts (ECMWF):

- P , the pressure at surface level.
- T , the temperature at 2 m.
- V_x , the x wind component at surface level.
- V_y , the y wind component at surface level.
- V , the wind norm at surface level.
- V_x^{100} , the x wind component at 100 m.
- V_y^{100} , the y wind component at 100 m.
- V^{100} , the wind norm at 100 m.

In the Sotavento case they are taken on a 15×9 rectangular grid with a 0.25° resolution centered on the Sotavento site (43.34°N , 7.86°W); input dimension in this case is thus $15 \times 9 \times 8 = 1080$. For peninsular Spain we consider a 57×35 rectangular grid that covers entirely the Iberian peninsula; input dimension is now a very large $57 \times 35 \times 8 = 15,960$. Wind energy data for Sotavento are publicly available; those for peninsular Spain were kindly provided by Red Eléctrica de España (REE). In both cases we normalize them to the $[0, 1]$ interval by dividing actual wind energy production by the maximum possible value in each case. We will work with data for the years 2011, 2012 and 2013, that we will use as training, validation and test subsets respectively. Since NWP forecasts are given every 3 h, each subset will approximately have $(24/3) * 365 = 2920$ patterns.

3.2 Building Deep Models

A key issue to achieve a good performance in DNN training is the correct choice of the several architecture and training hyper-parameters to be considered. However, the extremely large range of possibilities forces the practitioner to begin with a concrete approximation to the network structure before embarking on the very costly process of optimal hyper-parameter selection. In previous work ([10]) we considered for the wind energy problem several different deep network architectures, among which a convolutional architecture of the LeNet type [23] proved there to be the best choice. We will use an adaptation of the concrete architecture used in [10], which has

- Two initial convolutional layers, followed by
- Two fully connected layers and, finally,
- A final linear readout layer.

We will work here with non-symmetric ReLUs as hidden layer activations and apply the Glorot–Bengio weight initialization heuristic proposed in [13], with a 0-symmetric uniform weight distribution and an interval width (or, equivalently, uniform distribution variance) adjusted to the layers' fan-in. As in [10] we will refer to it as the LeNet-5 architecture.

We will consider each weather variable to define an input feature map which in the wind energy case implies that there 8 such input channels. This plus our overall DNN architecture leaves us with the following hyper-parameter ranges to be explored:

- Number of convolutional output channels: integers from 8 to 200.
- Number of fully connected hidden units: integers from 50 to 500, one per layer.
- Weight decay multipliers in fully connected layers: float from 0.0 to 0.5.

- Dropout fractions in hidden fully connected layers but not for the output weights: float from 0.1 to 0.9.
- Minibatch training size: integer from 50 to 500.
- Starting learning rate for the training algorithm: float from 1.0 to 0.00001 in a logarithmic scale.

Convolutional networks also require that strides and filter and pooling sizes are set. This would add further complexity to the hyper-parameter search and to avoid it here we set the stride to 1 and the filter size to 2×4 in the first convolutional layer and 3×5 for the second; we will not apply any kind of pooling. In any case, it is clear that the number of hyper-parameters is too large for an exhaustive grid search. To alleviate this we have used the Tree-structured Parzen Estimator approach available through the Hyperopt ([7]) library, with a maximum number of evaluations set to 200 that iteratively defines a random path in hyper-parameter space that progressively focuses on better values. Of the several training algorithms at our disposal (SGD, SGD–Nesterov, Adagrad and Adadelata), best results were obtained with Adadelata. We recall that it automatically adjusts its learning rate from an initial choice.

We selected the best hyper-parameter set by a simplified, time-structured validation procedure, using as our error measure over a concrete hyper-parameter set P the mean absolute error (MAE), i.e.,

$$MAE(P) = \frac{1}{N} \sum_{n=1}^N |D(x_n; P) - y_n|, \tag{9}$$

where $D(x; P)$ denotes the value on pattern x of the current deep network D built using the hyper-parameter set P . We use the MAE instead of the more often used squared error as it is the measure of choice in renewable energy, for it represents energy deviation and, thus, the energy to be shed or bought from other generation sources to compensate errors in energy estimates. For the above mentioned time structured validation, the year 2011 is used for training, 2012 for validation and 2013 for test; this is a quite natural choice when data have a strong temporal structure.

In Figure 1 we depict the evolution of the train and validation errors for the optimal Keras-based deep network we will describe in Sect. 3.4. While the training MAE present spikes due to the use of mini-batches, it decreases over the entire training. However, this decrease in training MAE does not result in the model overfitting; in fact validation MAE also present spikes but its value stabilizes even while training MAE keeps decreasing. Most likely this is due to the use of dropout. Because of this, and to shorten training times, our strategy is to set a maximum number of 1000 training epochs (i.e., passes through the entire training set), keep track of the best validation MAE after each iteration and to stop training if no improvement in validation MAE is achieved in the last 100 epochs.

3.3 Ensembles

As it is well known, ensemble learning is an important area of Machine Learning in which several machines or experts are combined to create a more accurate one. The concepts that support, in principle, the success of ensemble learning can be resumed in two simple ideas: the easiness of designing individual experts with good enough outputs (i.e., low bias), and the independent and random differences between the experts' output (that lowers variance). If these two requirements are satisfied, it is clear that with an appropriate aggregation of the experts' outcomes, the final output can have a higher quality [26].

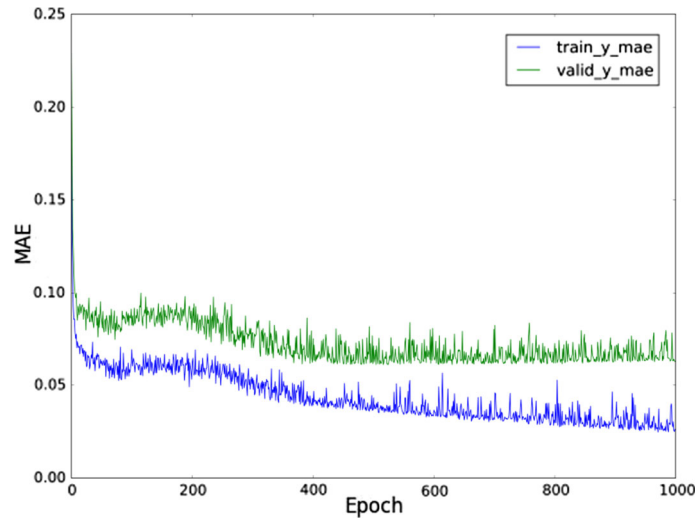


Fig. 1 Training and validation evolution of the optimal LeNet-5 network for Sotavento

Table 1 Mean absolute errors for the Sotavento and REE problems as reported in [10]

	MAE Sotavento			MAE REE		
	Test	Validation	Train	Test	Validation	Train
SVR	7.80	6.73	5.62	3.13	3.30	1.01
LeNet-5	7.63	6.25	5.82	3.13	3.01	2.48

There are many techniques to achieve diversity of learners in regression ensembles, such as bagging [8] in which each learner is trained on a subset of the original training dataset sampled with replacement, and random initialization in which all experts see all the training dataset but are initialized using different random seeds. These ideas aim to incorporate randomness during training, both on the training samples and, also and if possible, on the training procedure. Notice that Deep Networks exploit both ideas naturally: minibatch training ensures randomness on the training sample at each epoch, and random weight initialization, minibatch training again and dropout enforce randomness during model building. As we shall see in our experiments, ensembles of these networks are likely to improve on the performance of single models.

3.4 Wind Energy Results

As a starting point we recall the results in [10] in Table 1. While Pylearn-Theano was used in [10], here we will use the Theano/TensorFlow Keras wrapper for DNN training. Because of this we have re-trained DNNs for both Sotavento and REE. However, for the REE problem, Keras' based DNNs have not produced improvements on the results in [10] as given in Table 1; because of this we will report the new Keras-based results only for Sotavento. The previously described hyper-parameter search with Hyperopt yielded the following optimal hyper-parameters:

- 64 convolutional feature maps (channels) in the first layer and 128 in the second.
- Two fully connected 200 unit layers.

Table 2 SVR, Keras–LeNet and ensemble Mean Absolute Error for the Sotavento problem

	MAE Sotavento		
	Test	Validation	Train
SVR	7.80	6.73	5.62
Keras–LeNet	7.60	6.03	5.10
Ensemble	7.53	5.96	5.02

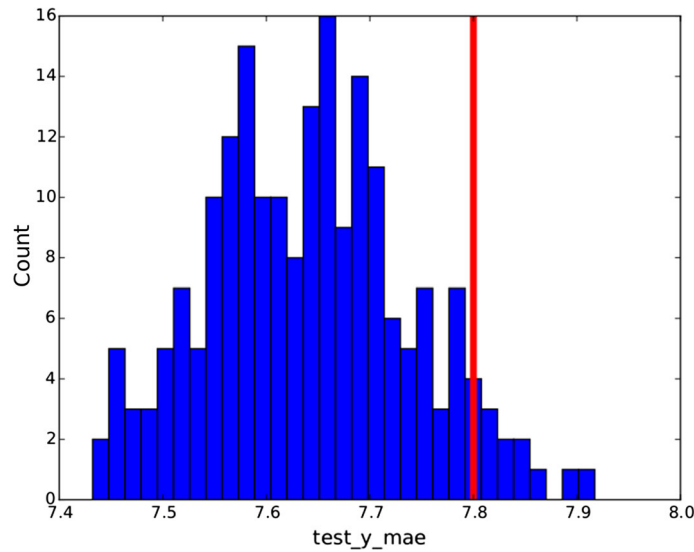


Fig. 2 Histogram of test errors over the individual 200 ensemble networks for the Sotavento problem

- A dropout coefficient of 0.2.
- A mini-batch size of 70.
- A learning rate of 0.3.

We point out that the optimal weight decay was extremely small in all cases (probably because of the regularizing effect of dropout), so at the end no weight decay was used. Moreover, recall that we preset strides to 1, convolutional filter sizes to 2×4 in the first layer and 3×5 in the second, and that no pooling will be applied. We will refer to this architecture as Keras–LeNet.

Table 2 gives training, validation and test errors for the optimal models with our new approach; we also include SVR for comparison. Notice that the new Keras–LeNet model slightly improves the results in [10] and that the DNN ensemble yields a noticeably better error rate. The ensemble MAE values have been obtained training 200 Keras–LeNet networks and averaging the predictions of those that yielded the smallest 25% validation errors

Figure 2 shows the histogram of the test error resulting from the 200 trained Keras–LeNet models; the red line represents the SVR test error. Notice that 94% of Keras–LeNet errors are below SVR error and that the best network (which, of course, cannot be identified beforehand) would have yielded a MAE of 7.43, rather close to the ensemble MAE. All this suggests that DNN ensembles are a robust way to improve single DNN performance.

4 Solar Radiation Experiments

In this section we will apply DNNs to the problem of predicting daily aggregated solar radiation for Oklahoma's Mesonet stations. We will also consider here each weather variable to define an input feature map; there will now be 15 input maps.

4.1 Solar Radiation Data

The NWP variables provided in the Kaggle competition were the following:

- *apcp_{sfc}*, 3-h accumulated precipitation at the surface.
- *dlwrf_{sfc}*, downward long-wave radiative flux average at the surface.
- *dsurf_{sfc}*, downward short-wave radiative flux average at the surface.
- *pres_{msl}*, air pressure at mean sea level.
- *pwat_{atm}*, precipitable water over the entire depth of the atmosphere.
- *spfh_{2m}*, specific humidity at 2 m above ground.
- *tcdc_{atm}*, total cloud cover over the entire depth of the atmosphere.
- *tc_{colc_{atm}}*, total column-integrated condensate over the entire atmosphere.
- *tmax_{2m}*, maximum temperature over the past 3 h at 2 m above the ground.
- *tmin_{2m}*, minimum temperature over the past 3 h at 2 m above the ground.
- *tmp_{2m}*, current temperature at 2 m above the ground.
- *tmp_{sfc}*, temperature of the surface.
- *ulwrf_{sfc}*, upward long-wave radiation at the surface.
- *ulwrf_{1atm}*, upward long-wave radiation at the top of the atmosphere.
- *uswrf_{sfc}*, upward short-wave radiation at the surface.

Here the grid had 9×16 points with 0.5° resolution. Recall that the targets were daily aggregated measurements of incoming solar radiation at the 98 stations of Oklahoma's Mesonet network. There were 5 forecasts available per day, from 1994 to 2007, corresponding essentially to sunlight hours at 3-h interval. NWP forecasts were available also for 2008 and 2009 but not the radiation measures; these years were thus provided by Kaggle as the test datasets. While NWP forecasts were given for a 11 member ensemble, we will only work with the first ensemble. Thus, the input dimension of the problem is $9 \times 16 \times 15 \times 5 = 10,800$. In our experiments we will use 1994 to 2005 as training dataset, 2006 as validation dataset, and 2007 as test dataset. The number of training patterns is thus essentially $365 \times 12 = 4380$.

4.2 Results for Solar Radiation

In the light of the results for the wind energy problem, we will also consider here only Keras–LeNet networks. Applying a new hyperopt-based hyper-parameter search over the same ranges used for wind energy, we have now obtained the following optimal set:

- A first convolutional layer with 150 output channels and a second one with 150 channels. As before, strides are set to 1, we use 2×5 filters in the first layer, 3×5 filters in the second and no pooling.
- Two fully connected 400 unit layers.
- Dropout coefficient of 0.2.
- Mini-batch size of 150.
- Starting learning rate of 0.3.

Again, we didn't use weight decay here and used Adadelta as the training algorithm. The solar energy problem is more demanding in terms of computational resources than the Sotavento

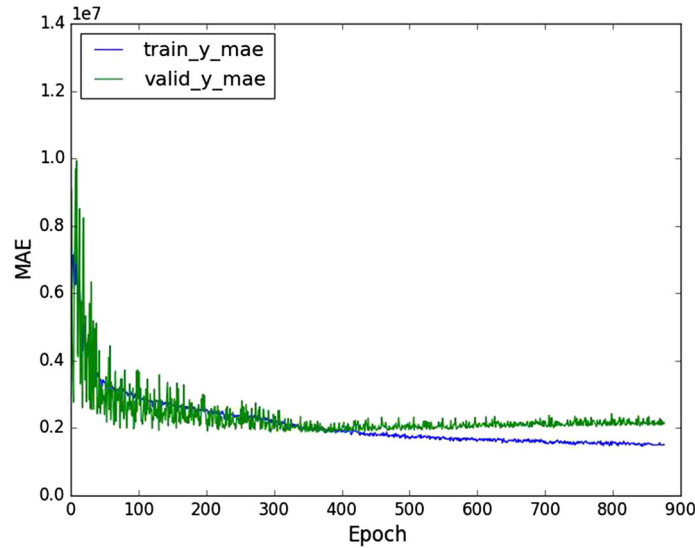


Fig. 3 Training and validation evolution of the optimal Keras–LeNet network for Solar

Table 3 SVR, Keras–LeNet and ensemble Mean Absolute Error for the Solar problem

	MAE Solar		
	Test	Validation	Train
SVR	2,225,252.24	1,917,895.20	1,417,140.20
Keras–LeNet	2,163,321.97	1,837,780.91	1,772,189.87
Ensemble	2,090,959.30	1,772,383.39	1,739,860.07

problem, so we have to decrease the number of networks per ensemble. We have decided to train 40 Keras–LeNet nets with the above optimal parameters and to select again those yielding the top 25% validation errors. For comparison purposes, we also consider a Gaussian SVR model whose C , γ and ϵ hyper-parameters have been established by a grid search over data from the first Mesonet station; their optimal values were $C=8,388,608$, $\gamma = 6.1035e - 05$ and $\epsilon = 4096$.

We point out that here again validation MAE values stabilize during training even if training errors keep decreasing. This is illustrated in Fig. 3, which shows the evolution of the train and validation errors for the optimal Keras–LeNet network; again, the regularizations used helps to avoid overfitting.

Table 3 displays the training, validation and test errors for the SVR, a Keras–LeNet single network and the Keras–LeNet ensemble. These have been computed by training one SVR model for each Mesonet station in the first case, while the DNN-based models have been trained over the data of all the stations at the same time in a multi-target regression configuration, working with a 98-dimensional target vector made up with the daily aggregated radiation for each station. Here again, we achieve a lower error using an ensemble. Our procedure makes clear that these models have not been built in order to compete with the best ones in the Kaggle contest. Among other things, we do not consider the entire set of NWP predictions available nor seek to build optimal DNN models for each one of the 98 Mesonet stations. At best, our models would be the core of a first submission to be improved

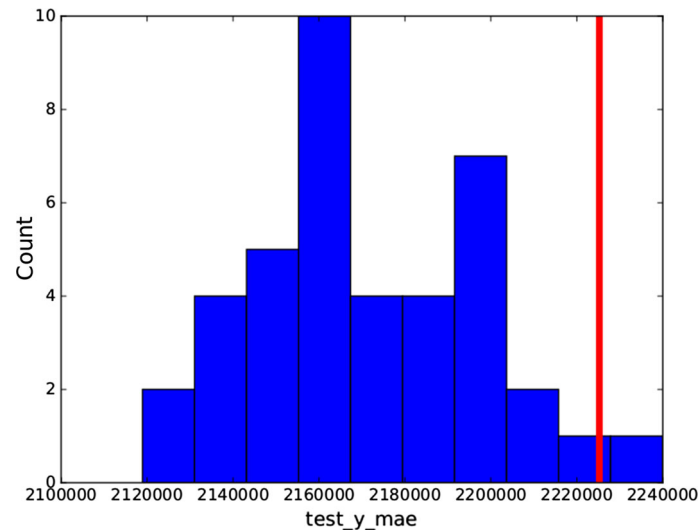


Fig. 4 Histogram of test errors over the individual 40 ensemble networks for the solar radiation problem

on subsequent ones (the 10 top teams made an average of 85 submissions each). Nevertheless, it is interesting to place our results in the competition's context and our single deep net model would have been placed in the 42-th position of the Private Leaderboard, with a MAE of 2,371,143.10. The ensemble brings a slight improvement, with a MAE of 2,365,637.10 and would occupy the 41-th position. The SVR model would be ranked in the 110-th position, with a MAE of 2,561,382.58.

We finally point out that ensemble training also seems to be quite robust here. Figure 4 shows the histogram of the test errors derived from the 40 Keras–LeNet models trained independently with the same hyper-parameter set. Again, the red line represents the SVR test error, which is even more in the right tail of the ensemble's MAE distribution that was the case for wind energy.

5 Conclusions

Deep networks are undeniably very powerful but also very costly to set up and train. However, this considerable training effort usually pays off as deep nets often produce better results than other classical models. As we have shown, this has been the case for the Sotavento wind energy prediction and the Kaggle data set for solar radiation from Oklahoma's Mesonet network; on the other hand, they essentially tied with SVRs for the wind energy prediction over peninsular Spain. Observe that in all cases the use of grid-based weather forecasts as inputs gives to both problems a bi-dimensional pattern structure; moreover, each individual variable can be naturally seen as an input channel to be processed by a convolutional layer.

As in many problems, a natural option to reduce variance is to build an ensemble that combines several deep models. We have pointed out how Deep Networks introduce and exploit independent randomness in a natural way by using random minibatch training, weight initialization and dropout regularization and our experiments show that DNN ensemble models can be quite robust and significantly improve the accuracy of a single network. Therefore,

deep networks are clear candidates to noticeably benefit from ensemble methods and this is part of our current research. Other research venue is the exploitation of some of the many new proposals for network initialization and architectures as well as model training and regularization that appear almost constantly from the ongoing great research effort in Deep Networks.

Finally, a weak spot of DNNs is the high cost of their training and, hence, model selection. The way out of this is to exploit the constant advances to, first, speed up single network training through the use of GPUs and, second, to shorten ensemble model building through parallelization. We are also studying some of the new proposals on these directions which are appearing almost continually, in particular the recent extensions of Google's TensorFlow library.

Acknowledgements With partial support from Spain's Grants TIN2013-42351-P (MINECO), S2013/ICE-2845 CASI-CAM-CM (Comunidad de Madrid), FACIL (Ayudas Fundación BBVA a Equipos de Investigación Científica 2016) and the UAM-ADIC Chair for Data Science and Machine Learning. The second author is also kindly supported by the FPU-MEC Grant AP-2012-5163. The authors gratefully acknowledge access to the MARS repository granted by the ECMWF, the use of the facilities of Centro de Computación Científica (CCC) at UAM and thank Red Eléctrica de España for kindly supplying wind energy production data and to Sotavento for making their production data publicly available.

References

1. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow IJ, Harp A, Irving G, Isard M, Jia Y, Józefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray DG, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker PA, Vanhoucke V, Vasudevan V, Viégas FB, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2016) Tensorflow: large-scale machine learning on heterogeneous distributed systems. CoRR abs/1603.04467. <http://arxiv.org/abs/1603.04467>
2. Arpit D, Zhou Y, Kota BU, Govindaraju V (2016) Normalization propagation: a parametric technique for removing internal covariate shift in deep networks. In: Proceedings of the 33rd international conference on machine learning, ICML 2016, New York City, NY, USA, June 19–24, 2016, pp 1168–1176. <http://jmlr.org/proceedings/papers/v48/arpitb16.html>
3. Bastien F, Lamblin P, Pascanu R, Bergstra J, Goodfellow IJ, Bergeron A, Bouchard N, Bengio Y (2012) Theano: new features and speed improvements. In: Deep learning and unsupervised feature learning NIPS 2012 workshop
4. Bengio Y, Courville AC, Vincent P (2013) Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828. doi:10.1109/TPAMI.2013.50
5. Bengio Y, Lamblin P, Popovici D, Larochelle H (2007) Greedy layer-wise training of deep networks. In: Advances in neural information processing systems 19 (NIPS'06), pp 153–160. <http://www.iro.umontreal.ca/~lisa/pointeurs/BengioNips2006All.pdf>
6. Bergstra J, Breuleux O, Bastien F, Lamblin P, Pascanu R, Desjardins G, Turian J, Warde-Farley D, Bengio Y (2010) Theano: a CPU and GPU math expression compiler. In: Proceedings of the Python for Scientific Computing Conference (SciPy). Oral Presentation
7. Bergstra J, Yamins D, Cox DD (2013) Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: Proceedings of the 30th international conference on machine learning
8. Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140
9. Chollet F (2015) Keras: deep learning library for theano and tensorflow. <http://keras.io>
10. Díaz D, Torres A, Dorronsoro JR (2015) Deep neural networks for wind energy prediction. In: Advances in computational intelligence—13th international work-conference on artificial neural networks, IWANN 2015, Palma de Mallorca, Spain, June 10–12, 2015. Proceedings, Part I, pp 430–443
11. Duchi JC, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12:2121–2159. <http://dl.acm.org/citation.cfm?id=2021069>
12. E.C. for Medium-Range Weather Forecasts: European center for medium-range weather forecasts. <http://www.ecmwf.int/>

13. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: JMLR W&CP: proceedings of the thirteenth international conference on artificial intelligence and statistics (AISTATS 2010), vol 9, pp 249–256
14. Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks. In: JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)
15. Goodfellow IJ, Warde-Farley D, Lamblin P, Dumoulin V, Mirza M, Pascanu R, Bergstra J, Bastien F, Bengio Y (2013) Pylearn2: a machine learning research library. arXiv preprint [arXiv:1308.4214](http://arxiv.org/abs/1308.4214). <http://arxiv.org/abs/1308.4214>
16. He K, Zhang X, Ren S, Sun J (2015) Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CoRR abs/1502.01852. <http://arxiv.org/abs/1502.01852>
17. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507. doi:10.1126/science.1127647. <http://www.sciencemag.org/content/313/5786/504.abstract>
18. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd international conference on machine learning, ICML 2015, Lille, France, 6–11 July 2015, pp 448–456. <http://jmlr.org/proceedings/papers/v37/loff15.html>
19. Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick RB, Guadarrama S, Darrell T (2014) Caffe: convolutional architecture for fast feature embedding. CoRR abs/1408.5093. <http://arxiv.org/abs/1408.5093>
20. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. CoRR abs/1412.6980. <http://arxiv.org/abs/1412.6980>
21. Kruger N, Janssen P, Kalkan S, Lappe M, Leonardis A, Piater J, Rodriguez-Sanchez A, Wiskott L (2013) Deep hierarchies in the primate visual cortex: what can we learn for computer vision? *IEEE Trans Pattern Anal Mach Intell* 35(8):1847–1871. doi:10.1109/TPAMI.2012.272
22. Lecun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436–444. doi:10.1038/nature14539
23. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. In: Proceedings of the IEEE, vol 86, no 11, pp 2278–2324
24. LeCun Y, Bottou L, Orr G, Muller K (1998) Efficient backprop. In: Orr G, K M (eds) *Neural networks: tricks of the trade*. Springer, Berlin
25. Ma H, Mao F, Taylor GW (2016) Theano-mpi: a theano-based distributed training framework. CoRR abs/1605.08325. <http://arxiv.org/abs/1605.08325>
26. Murphy KP (2012) *Machine learning: a probabilistic perspective*. MIT Press, Cambridge
27. NOAA: Global forecast system. <http://www.emc.ncep.noaa.gov/index.php?branch=gfs>. <http://www.emc.ncep.noaa.gov/index.php?branch=GFS>
28. Society AM (2013) 2013–2014 solar energy prediction contest. <https://www.kaggle.com/c/ams-2014-solar-energy-prediction-contest>
29. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. *J Mach Learn Res* 15: 1929–1958. <http://jmlr.org/papers/v15/srivastava14a.html>
30. Sutskever I, Martens J, Dahl GE, Hinton GE (2013) On the importance of initialization and momentum in deep learning. In: Dasgupta S, Mcallester D (eds) Proceedings of the 30th international conference on machine learning (ICML-13), vol 28, pp 1139–1147. JMLR workshop and conference proceedings. <http://jmlr.org/proceedings/papers/v28/sutskever13.pdf>
31. Zeiler MD (2012) ADADELTA: an adaptive learning rate method. CoRR abs/1212.5701. <http://arxiv.org/abs/1212.5701>

Deep Neural Networks for Wind Energy Prediction

David Díaz^(*), Alberto Torres, and José Ramón Dorronsoro

Departamento de Ingeniería Informática e Instituto de Ingeniería del Conocimiento,
Universidad Autónoma de Madrid, Madrid, Spain
david.diazv@estudiante.uam.es, {alberto.torres,jose.dorronsoro}@uam.es

Abstract. In this work we will apply some of the Deep Learning models that are currently obtaining state of the art results in several machine learning problems to the prediction of wind energy production. In particular, we will consider both deep, fully connected multilayer perceptrons with appropriate weight initialization, and also convolutional neural networks that can take advantage of the spatial and feature structure of the numerical weather prediction patterns. We will also explore the effects of regularization techniques such as dropout or weight decay and consider how to select the final predictive deep models after analyzing their training evolution.

1 Introduction

Having had a big first impact around 1990, Multilayer Perceptrons (MLPs) started a mild decline after the second half of that decade. A particularly puzzling issue was the difficulty to build efficient MLPs with three or more layers, in spite of the fact that the backpropagation computation of the gradient of the MLP error function could be carried out in a rather straightforward fashion. The reason behind this was the vanishing gradient phenomenon [8] which in turn was in part a consequence of the inadequacy of weight initialization.

However, this changed radically with the seminal paper by G. Hinton and R. Salakhutdinov [12] that showed how an unsupervised, stacking scheme based on Boltzmann machines could yield a good initialization (or pretraining) of the weights of a many-layered MLP, that could be then efficiently fine-tuned by backpropagation. Shortly afterwards, Y. Bengio and his coworkers proposed a similar and somewhat simpler pretraining using stacked autoencoders [4]. This opened the way to the enormous attention that deep MLPs (i.e., MLPs with three or more layers) or, in general, deep learning, have received in the past years.

This attention has in turn resulted in a great simplification of the initial schemes of Hinton and Bengio and has brought many new procedures and ideas to the MLP field, such as new initializations, or the replacement of some of the initial MLP recipes, such as sigmoid activations or weight decay regularization, by new proposals like rectified linear unit (ReLU) activations [9] or dropout

regularization [18]. Moreover, the very large datasets and deep MLP parameters often rule out batch learning. This has resulted in a large emphasis on online learning, usually over minibatches of randomly selected patterns, with much work being devoted to the choice of learning rates (or how to avoid them) or momentum methods such as Nesterov's acceleration. This raises the issue of when to stop training, something rather straightforward in the batch training of classical MLPs if an adequate regularization and an efficient optimizer were used. Furthermore, once the previous ingredients are in place, the need to specialized (and costly) pretraining is less acute and several initialization methods have been proposed that result in the training of effective deep models. A good example of such a global approach is [19]. Another key ingredient in the successful applications of deep learning is the use of convolutional layers, that concatenate a purely convolutional sublayer that processes inputs using localized window filters, and a pooling sublayer that aggregates the outputs of the previous sublayer. Starting with the work of Y. LeCun in the late 1990's, this processing is particularly natural when inputs have a spatial structure, as it is the case with images, and it has led convolutional deep nets to achieve state-of-the-art results in problems such as MNIST [7] or ImageNet [14].

All these advances have made possible the effective training of very large deep networks with hundreds of thousands of weights which, in turn, makes imperative the use of software that can take advantage of high performance hardware endowed with parallelization (i.e., multicore machines) and vectorization (i.e., GPU units). Besides, the fast pace of change in the field and the fact that there is still not an accepted multipurpose architecture makes it quite difficult to work with self developed code; instead, it is well advised to rely on publicly available libraries and environments such as the Caffe [13] deep learning framework or the Pylearn2–Theano libraries [5] [3] [10] that we use here.

In any case, it seems that the bulk of deep learning research concentrates on computer vision, speech recognition and natural language processing problems. This is partially natural in view of the broad similarity between deep learning architectures and the processing hierarchies in the visual cortex [15] (although deep learning algorithms are quite different from the Spike-Timing-Dependent Plasticity learning rule most accepted in neurobiology). As such, deep learning algorithms are increasingly seen as representation learning procedures that yield at each layer increasingly more abstract representations in such a way that features in the higher layers capture possibly more powerful data features.

However, the successful exploitation of such a processing may also take place in simpler regression problems that, nevertheless, have input patterns with a spatial structure. The goal of this work is the prediction of wind energy production. Spain is among the world leaders in wind energy with a very high penetration that in some special days and hours can meet a very high percentage of Spain's electricity demand. Obviously, this high penetration makes it very important to provide accurate prediction of wind energy, with standard MLPs (usually at the farm level) and Support Vector Regression (SVR) (for large scale prediction) being the models of choice. The inputs for such models are the forecasts provided

by numerical weather prediction (NWP) systems such as the ECMWF [1] or the GFS [2]. These predictions are forecasts of several weather variables given at the points of a rectangular grid that covers the areas under study and that reflects some underlying orographic model. One may thus view an area wide NWP forecasts as a set of feature maps (the individual weather variables) having a spatial structure (that of the underlying geography) in much the same way that the RGB channels of an image correspond to feature maps with a two dimensional structure.

Under the previous scheme, the consideration of convolutional networks to derive wind energy forecast arises as a natural option and they will be one of the models considered in this paper. Our main purpose is to develop a methodology to build models that can provide accurate predictions from the original data with as little pre-processing and expert knowledge as possible. Given the very wide range of proposals in the literature, this implies we must make beforehand concrete choices of network initialization, online training procedure, activation function and regularization scheme. Of course, on top of this, a more or less general network architecture also has to be selected.

We will develop the choices we make in the next sections. Besides standard “small” MLPs and SVR models that we use as reference benchmarks, we will consider deep MLPs with a standard multilayer structure, general deep convolutional networks (CNNs) and also an adaptation of the well-known LeNet [16], one of the most successful architectures for character recognition. In all those deep nets we will use Glorot–Bengio weight initialization [8], ReLUs as activation functions [9], dropout regularization [18] complemented with standard weight decay in the final fully-connected layers, random mini-batch gradient descent over batches of moderate size and conjugate gradient as the training algorithm. This enables us to work with a fixed, fairly general learning rate, that is no longer a parameter to explore. Summing things up, our main contributions are:

- We review some of the latest proposals in DNNs and propose general guidelines to apply deep MLPs in regression problems.
- We thoroughly explore the application of the two main paradigms in DNNs to the problem of local and large scale wind energy prediction.
- We introduce a variant of the well known LeNet convolutional neural network adapted to wind energy prediction and show it to be very competitive with other DNN architectures or state of the art methods such as Support Vector Regression.

As mentioned before, we will use Pylearn2 [10]–Theano [3] [5] platform as it includes a wide variety of already tested neural networks and allows us to explore several of the latest and most effective proposals for deep network training. An important advantage of having Theano as the underlying numerical library is that we can exploit its capabilities for code execution on GPUs, something crucial given the network sizes and input dimensions we work with. We run our experiments on a machine equipped with a NVidia Tesla K40 GPU which makes possible reasonable execution times and, hence, the capability of exploring a fairly large number of deep model configurations.

The rest of the paper is organized as follows. In Section 2 we review our choices for deep network configuration and discuss some of its details. Section 3 contains a succinct description of the framework for wind energy prediction over NWP inputs, a description of our experimental setup and the prediction results for both the Sotavento wind farm and the entire wind energy prediction over peninsular Spain that is overseen by Red Eléctrica de España (REE). Finally, in Section 4 we briefly discuss our results and offer pointers to further work.

2 Deep Neural Networks

We briefly review here some of the key issues when configuring and training Deep Neural Networks.

2.1 Initialization

There have been several heuristic proposals for weight initialization in “classical” MLPs. For instance, a common choice is to take them from a uniform distribution $U \left[-\frac{1}{\sqrt{M}}, \frac{1}{\sqrt{M}} \right]$, with M the fan-in of the neuron, i.e., the number of weights feeding into it. However, it was found experimentally in [8] that in a deep MLP initialized in such a way, back-propagated gradients were progressively smaller when moving from the output layer towards the input layer and, in addition, their variances also decrease. In other words, backpropagating such an initialization may result in vanishing gradients in the first layers following the input and, thus, in a network which is insensitive to its inputs and unable to “learn” them.

The more detailed analysis in [17], also oriented to “classical” MLPs and where properly normalized hyperbolic tangents were used, pursued a goal of keeping the (linear) activations and (non linear) outputs of a neuron in the $[-1, 1]$ active range of the (normalized) hyperbolic tangent. Assuming inputs normalized to zero mean and unit variance component-wise, it is suggested in [17] to use an uniform distribution $U \left[-\frac{\sqrt{3}}{\sqrt{M}}, \frac{\sqrt{3}}{\sqrt{M}} \right]$. This analysis was extended in [8], where, assuming again the neuron outputs z_i of the i -th layer also to be in the $[-1, 1]$ active range, initialization should ensure first that $Var(z_i) \simeq Var(z'_i)$ across the successive layers and also that $Var \left(\frac{\partial J}{\partial z_i} \right) \simeq Var \left(\frac{\partial J}{\partial z'_i} \right)$, where J denotes the MLP cost function. This translates into the following equations for the initial weights W_i

$$M_i Var(W_i) = 1; M_{i+1} Var(W_i) = 1$$

where M_i and M_{i+1} are the fan-in and fan-out of the units in the i -th layer. An approximation to both is to take $Var(W_i) = \frac{2}{M_i + M_{i+1}}$, i.e., to initialize the W_i using an uniform distribution $U \left[-\frac{\sqrt{6}}{\sqrt{M_i + M_{i+1}}}, \frac{\sqrt{6}}{\sqrt{M_i + M_{i+1}}} \right]$. Note that when $M_i = M_{i+1}$, we get back the initialization proposed in [17].

We will use the initialization in [8] but working with Rectified Linear Unit (ReLU) activations, discussed next, instead of the hyperbolic tangent ones. While the rationale in [8] may not apply, the recent analysis in [11] of weight initialization for ReLU activation suggests to dilate the Glorot–Bengio uniform intervals by a factor of 1.5, and, in fact, we have observed that this usually yields better results.

2.2 Activation Function

As mentioned, we have used ReLUs for all hidden layer activations and linear units in the output layer. The ReLU transfer function is $r(x) = \max(0, x)$, that is, their response to the opposite of a positive excitatory input is just 0; in particular, ReLUs do not have a sign antisymmetry, as is the case with the hyperbolic tangent. On the other hand, ReLUs share some similarities with the functions relating neuronal input currents and firing rates that appear in the leaky integrate and fire models used in biological neuron models [9]. Besides, ReLUs induce sparsity in the representations of the successive layers; for instance, right after the uniform weight initialization, the outputs of about half the network neurons should be zero, as they would correspond to negative (inhibitory) inputs. This may partially explain the fact that ReLUs seem to be less affected than other activations by poor initializations. In any case, this point deserves further study.

2.3 Regularization

It is obvious that the extremely large number of weights in a deep MLP makes regularization mandatory to avoid overfitting. The standard regularization technique in classical MLPs is weight decay applied across all the layers; i.e., the square norm weight penalty considered for all layer weights is added to the MLP cost function. It has an obvious place in a last layer with linear outputs, as it performs ridge regression on the features induced in that last layer by the deep processing of the inputs.

However, for the other layers we will use dropout [18], that we briefly describe next. If a_i^l denotes the i -th activation of the l -layer and z_i^l the corresponding output, the standard feedforward processing would yield $z_i^l = f(a_i^l) = f(w_i^l z^{l-1} + b_i^l)$, where f is the activation function. However, with dropout, a 0–1 vector r^l is first generated applying a Bernoulli distribution componentwise. The feedforward process becomes

$$z_i^l = f(a_i^l) = f(w_i^l (z^{l-1} \odot r^l) + b_i^l),$$

where \odot denotes the componentwise product. Each element in r^l has a probability p of being 1, so dropout can be seen as sub-sampling a larger network at each layer. The output errors are backpropagated as in standard MLPs for gradient computations and the final optimal weights w^* are downscaled as $w_f^* = pw^*$ to yield the final weights used for testing.

Dropout clearly induces a regularization of the network’s weights. Moreover, it is reminiscent to the well known bagging technique for ensembles that repeatedly subsamples data to build specific models and then takes the average. However, in dropout all the “models” (i.e., the particular feedforward Bernoulli realizations) share weights and they are “trained” in a single step. Although we will not use it, in [18] it is also suggested that network performance improves when dropout is combined with a bound on the L_2 norm of the weights, i.e., when they are constrained as $\|w\|_2 \leq c$, with c a second tunable parameter on top of the Bernoulli probability p .

2.4 Convolutional Layers

Standard deep MLP architectures tend to favor layers with a high number of hidden units. This also leads to a high number of weights, $M \times M'$ if we fully connect an M unit layer with an M' one, a number that can become rather large if, for instance, inputs have a two dimensional structure, as it is the case with images. Convolutional layers arise in part as a way to avoid this by limiting the fan-in of a hidden unit to come from a localized subset of units in the previous layer. Of course, how to define such a restricted fan-in is, in general, problem-dependent, but when data have an intrinsic spatial structure a natural approach to localize the connections is to work over small patches.

More precisely, assume inputs or layer outputs to be one channel structured as an $M_1 \times M_2$ matrix, and consider in them $K \times K$ submatrix patches. They could be either disjoint or partially overlapping; we can parameterize this considering a stride value S that gives the displacement applied when we move horizontally and vertically from one patch to the next one. Assuming for simplicity a $S = 1$ stride, there are such $(M_1 - K + 1) \times (M_2 - K + 1)$ (overlapping) patches x_j . A first transform is to derive a patch feature $p_j = f(w * x_j + b)$ where f is the activation function, $*$ denotes the convolution operator between the $K \times K$ filter w and the patch x_j and b is the bias of the filter. This transforms an $M_1 \times M_2$ input X into an $(M_1 - K + 1) \times (M_2 - K + 1)$ convolutional output X' and usually a number L of filter pairs (w_l, b_l) (or of feature maps) have to be learned. Thus, the number of weights in a convolutional sub-layer is a rather modest $L \times K^2$ but, on the other hand, the output dimension would be $L \times (M_1 - K + 1) \times (M_2 - K + 1)$, which for $L > K$ might greatly increase the number of hidden units in the next layer. To curb this (and avoid a possible overfitting), a second pooling (or subsampling) sub-layer is applied in which an operation such as averaging or computing the max is applied on $P \times P$ patches of X' to derive the final output X_C of the convolution–pooling combined process; X_C has a $L \times (M_1 - K - P + 2) \times (M_2 - K - P + 2)$ dimension.

This combined convolution–pooling process is called a convolutional layer; it allows for a localized processing of the layer’s input using a moderate number of weights (note that there are no weights in the pooling sub-layer) while arriving at a number of units in the next layer similar to that of the previous one. Of course, we stress again that, to be effective, a convolutional layer must act on inputs that have a spatial structure (such as images) and are naturally distributed in

feature channels (such as the RGB decomposition). This is also happens in our case, where weather prediction has an obvious spatial structure in which different meteorological features (pressure, temperature, wind components, etc.) can be seen as corresponding to different channels.

3 Experiments

In this section we will apply DNNs to the problem of predicting wind energy production, first on the Sotavento wind farm and then over peninsular Spain.

3.1 NWP and Production Data

We will work with the following eight meteorological variable forecasts given by the European Centre for Medium-Range Weather Forecasts (ECMWF) system for Numerical Weather Prediction (NWP):

- P , the pressure at surface level.
- T , the temperature at 2m.
- V_x , the x wind component at surface level.
- V_y , the y wind component at surface level.
- V , the wind norm at surface level.
- V_x^{100} , the x wind component at 100m.
- V_y^{100} , the y wind component at 100m.
- V^{100} , the wind norm at 100m.

In the Sotavento case they are taken on 15×9 rectangular grid centered on the Sotavento site (43.34°N , 7.86°W); input dimension in this case is thus $15 \times 9 \times 8 = 1,080$. For peninsular Spain we consider a 57×35 rectangular grid that covers entirely the Iberian peninsula; input dimension is now a very large $57 \times 35 \times 8 = 15,960$.

Wind energy data for Sotavento are publicly available; those for peninsular Spain were kindly provided by Red Eléctrica de España (REE). In both cases we normalize them to the $[0, 1]$ interval by dividing actual wind energy production by the maximum possible value in each case. We will work with data for the years 2011, 2012 and 2013, that we will use as training, validation and test subsets respectively. Since NWP forecasts are given every three hours, each subset will approximately have $(24/3) * 365 = 2,920$ patterns.

3.2 Deep Models

We will consider deep networks with either all their layers being fully connected, which we call deep MLPs, or with a number of initial convolutional layers followed by fully connected ones; we call these models deep convolutional neural networks, or deep CNNs. As reference models we will work with “standard” one hidden layer MLPs and also with Support Vector Regression (SVR) models,

Algorithm 1 Hyper-parameter search

```

1: procedure HYPER-PARAMETER SEARCH( $n, m$ ) ▷  $n \times m$  iterations
2:   randomly initialize an hyper-parameter vector  $p$ 
3:    $p^* = p$  ▷  $p^*$ : optimal hyper-parameter vector
4:   for  $i = 1, \dots, n$  do
5:     for  $j = 1, \dots, m$  do
6:        $k \leftarrow$  random value in  $\{1, \dots, m\}$ 
7:        $p_k \leftarrow$  random value in  $\{v_1^k, \dots, v_{N_k}^k\}$ 
8:       evaluate the  $p$ -parameterized model and update  $p^*$  if needed
9:     end for
10:  end for
11:  return  $p^*$ 
12: end procedure

```

among the most powerful modelling methods in wind energy prediction. The number of possible architectures and the many choices available for them would result in an unmanageable number of model hyperparameters to explore when looking for the best ones. To limit this, we have first fixed some of them to reasonable values that give good results in a first coarse model exploration.

A first such choice is that of the deep architectures to be considered. For deep MLPs we will consider two hidden layers with the same number of units. Our first choice for deep CNNs, which we call standard deep CNN or sdCNN, will have an initial convolutional layer followed by two fully connected layers again with the same number of units. Our second CNN choice, which we call LeNet CNN or lnCNN, will be an adaptation of the well known LeNet-5 architecture [16], that was specifically designed for the MNIST character recognition problem.

We will use the non-symmetric ReLUs at the hidden layers and, as discussed before, for network initialization we will apply the Glorot-Bengio heuristic proposed in [8] of using a 0-symmetric uniform distribution with a width adjusted to the layers' fan-in, scaling then up these initial weights by a factor of 1.5.

The training algorithm we are going to use for all the experiments is conjugate gradient descent (CGD) over random mini-batches. In other words, over each new mini-batch we apply CGD starting at the weights derived over the previous mini-batch; their size clearly affects the performance of the network and we have used sizes of either 200 or 250, i.e., about 6% and 9% of the training sample size. Our error measure is the mean absolute error (MAE)

$$MAE = \frac{1}{N} \sum_{n=1}^N |D(x_n; P) - y_n|,$$

where $D(x; P)$ denotes the value on pattern x of the current deep network D built using the hyperparameter set P . We use the MAE instead of the more often used squared error as it is the measure of choice in renewable energy, for it represents energy deviation and, thus, the energy to be shed or obtained from other generation sources to compensate errors in wind energy estimates.

As we shall see in the next subsection, the overall MAE evolution during training is decreasing but it often presents spikes due to the use of mini-batches, and this carries on to validation MAE values. In addition, validation MAE seems to stabilize even while training MAE keeps decreasing. Because of this, our model selection strategy is to train a deep NN while there is at least a 1% drop in MAE in the last 100 epochs, with a maximum of 1,000 epochs (i.e., goes through the entire training set). For convolutional networks we will consider each weather variable to define an input feature map; there are thus 8 such features. The above choices leave us with the following hyperparameters to be selected:

- For deep MLPs (which we denote by MLP2) we have to decide on the number (one or two) of hidden layers, the number of hidden units per layer, the weight decay and dropout coefficients, and mini-batch size.
- For the standard deep CNN (which we denote by CNN) we add to the previous deep MLP parameters the convolutional filter and pooling sizes, and their strides.
- For the LeNet CNN (which we denote by LeNet) we also have to decide on the deep MLP parameters but we simplify the other choices by selecting filter and pooling sizes and strides as adequately scaled versions to our problem of the choices made for LeNet-5.

In any case, it is clear that even after the previous simplifications, the number of hyperparameters is too large for an exhaustive grid search. To alleviate this we have used a greedy approach in which we fix first the number of fully connected hidden layers as 2 and then apply Algorithm 1, in which models are evaluated in terms of the MAE over the validation subset. The algorithm performs $n = 50$ external iterations on each of which a concrete hyper-parameter vector p is evaluated. The hyper-parameters considered are the number of hidden units in fully connected layers, the weight decay multipliers used in them, the dropout fraction and the minibatch size. On each external iteration m random choices are made of hyper-parameter indices k and for each a possible updating value p_k is randomly selected from the list $\{v_1^k, \dots, v_{N_k}^k\}$ of values of the k -th hyper-parameter to be explored. Both random selections are uniform. Actual tested values were

- Hidden unit numbers: 50, 100, 150, 200, 250, 300, 350, 400.
- Weight decay multipliers: 0.1, 0.2, 0.3, 0.4, 0.5.
- Dropout fractions: 0.3, 0.4, 0.5, 0.6, 0.7, 0.8.
- Minibatch size: 50, 100, 150, 200, 250, 300.

For the deep CNNs we fixed the stride to 1 and adjusted filter and pooling sizes by a limited heuristic search; notice that these sizes imply at least four more parameters and a fully random search over the entire parameter set is nearly impossible. The same is true for the number of convolutional feature maps. The just described hyperparameter search results in the following deep NN definitions:

Table 1. Mean Absolute Errors for the Sotavento and REE problems

	MAE Sotavento			MAE REE		
	Test	Validation	Train	Test	Validation	Train
SVR	7.80	6.73	5.62	3.13	3.30	1.01
LeNet	7.63	6.25	5.82	3.13	3.01	2.48
CNN	7.76	6.26	5.39	3.31	3.05	1.96
MLP2	7.76	6.33	5.86	3.37	2.96	1.97
MLP1	8.25	6.41	5.51	3.70	3.10	1.81

- Deep MLPs (MLP2) for Sotavento will have two hidden layers of 250 units, a weight decay coefficient of 0.3 and dropout coefficient of 0.7; mini-batch size is 200. The REE ones will have the same weight decay and dropout coefficients, two hidden layers of 300 units and mini-batch size is 250.
- Standard deep CNNs (CNN) for Sotavento will have a first convolutional layer with 2×6 filters and max pooling is performed over 2×2 patches. This layer is followed by two fully connected layers of 200 units, no weight decay, dropout coefficient of 0.7 and mini-batch size 250. The REE CNN has the same structure; the first layer has now 3×3 filters, max pooling is done over 3×5 patches. This is followed by two fully connected layers of 400 units, weight decay and dropout coefficients are 0.3 and 0.7 respectively and mini-batch size is 200. We used 16 convolutional feature maps for Sotavento and 8 for REE.
- The adapted LeNet-5 (LeNet) network for Sotavento has a first convolutional layer with 2×2 filters and max pooling, and a second one with 4×2 filters and 2×2 max pooling. They are followed by two fully connected 200 unit layers, no weight decay, dropout coefficient of 0.7 and mini-batch size 250. For REE, the LeNet network has a first convolutional layer with 6×8 filters, 2×2 max pooling, and a second one with 6×6 filters and 2×2 max pooling. They are followed by two fully connected 200 unit layers, weight decay and dropout coefficients of 0.3 and 0.7 respectively, and mini-batch size is 200. For both problems we used 16 convolutional feature maps in the first layer and 32 in the second.

3.3 Results

For comparison purposes, we also consider a Gaussian SVR model and a “standard” one-hidden layer, 10-unit MLP. We have used the very well known LIB-SVM library [6] and the SVR hyperparameters C , γ and ϵ have been established by a grid search; their optimal values were $C=128.0$, $\gamma = 3.0518 \times 10^{-5}$ and $\epsilon = 0.0625$ for Sotavento and $C=128.0$, $\gamma = 12.2078 \times 10^{-5}$ and $\epsilon = 0.01$ for REE. For the standard MLPs we used again Pylearn2-Theano and the optimal parameters were 0.001 weight decay coefficient and 200 mini-batch size in Sotavento and 0.1 weight decay coefficient and 250 mini-batch size for REE.

Table 2. Training complexity parameters and times in seconds for the Sotavento (top) and REE (bottom) deep models

Model	#Params.	#Iters.	Time	Time/Iter.
LeNet	140808	426	1175	2.76
CNN	105736	500	705	1.41
MLP2	332750	259	276	1.07
LeNet	224776	949	19494	20.54
CNN	548176	717	6880	9.60
MLP2	4878300	258	1208	4.68

Table 1 gives training, validation and test errors for the optimal models and the two problems. As it can be seen, the SVR and LeNet-5 models have a similar in performance in the REE problem, followed by the other two deep models; the standard MLP is in a distant last place. However, in Sotavento the LeNet-5 model is clearly the best model while the SVR and the two deep models essentially tie for second place; again, the standard MLP comes in last place. We point out that although we follow a straightforward train-validation-test scheme for model evaluation, a more accurate comparison should be made using an appropriate statistical test such as the well known Wilcoxon Rank Sum test, that takes into account not only MAE values but also standard deviations. This requires larger training periods and will be considered in further work.

Figure 1 shows the evolution of the train, test and validation errors for the optimal CNN and LeNet-5 networks for Sotavento (top) and REE (bottom). The large error variations are caused by mini-batch training; while at first sight validation and test error evolution appears smoother for Sotavento, this is partially due to a scale effect (about twice as large for Sotavento than for REE). For Sotavento the smallest errors seem to have essentially reached stable values; this is also the case for the validation and test errors in REE although training error would keep on decreasing, probably because the higher dimensionality of this problem. In both cases the vertical dotted line indicates the epoch with a lowest validation error and the horizontal dotted lines indicate the training, validation and test errors in that epoch. These are the values reported in Table 1.

Finally, in Table 2 we give the complexity parameters and training times in seconds for the deep models used in the Sotavento (top) and REE (bottom) problems. As it can be seen, all models are rather large, and more so those used for REE (remember that input dimensions are respectively 1,080 and 15,960). Besides, while the convolutional networks have less parameters than MLP2, their feedforward passes are much costlier due to the convolution operations and the same is true for the backpropagation of gradients. We observe that the smaller number of weights in LeNet for REE is due to the larger filters used.

It follows that deep training is rather costly and must take advantage of all possible hardware-based improvements available. In our case experiments

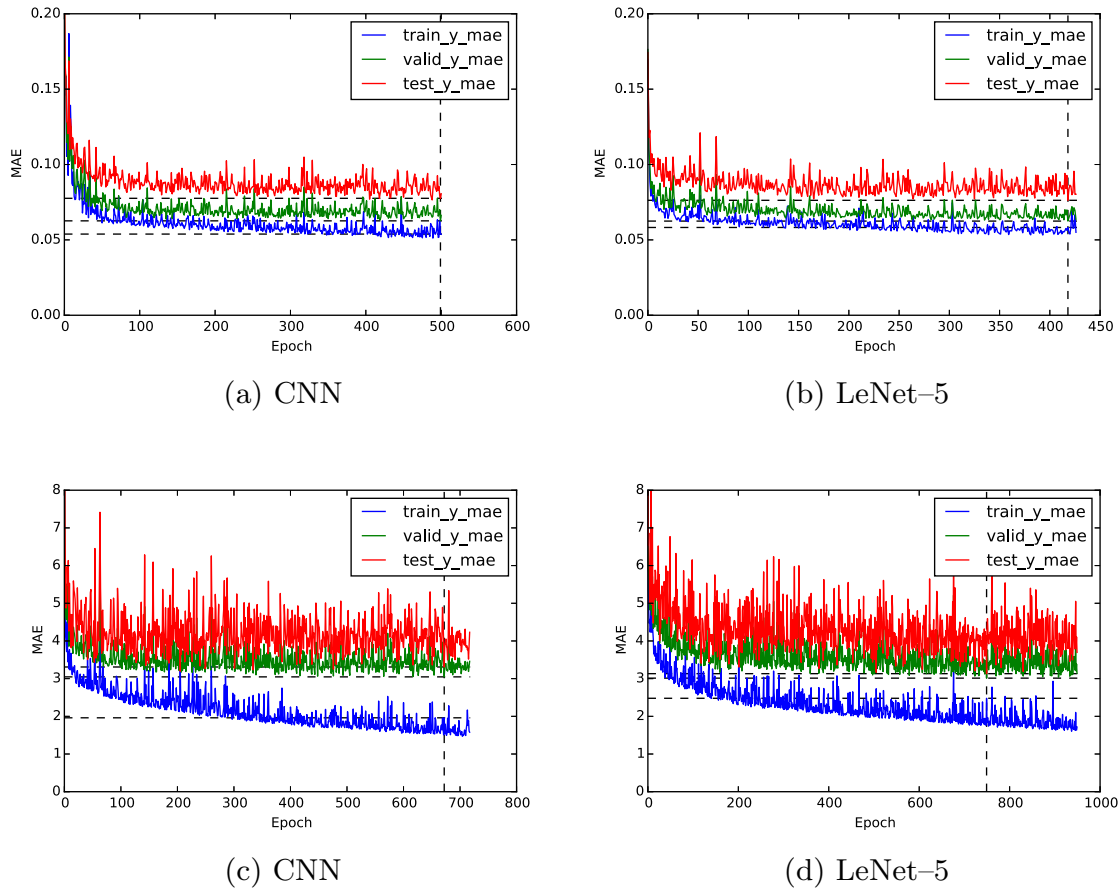


Fig. 1. Training, validation and test evolution of the optimal CNN and LeNet-5 networks for Sotavento (top) and REE (bottom)

have been run on a machine equipped with a NVidia Tesla K40 GPU and the Pylearn2-Theano framework. Working with Pylearn2 eases somewhat the development process, since most of it is written in Python.

There are other platform alternatives with the already mentioned Caffe being an interesting one, as its core is written in C++ and CUDA, which should result in a performance improvement. Another important improvement comes from the NVidia cuDNN library, that inter-operates perfectly with Pylearn2-Theano (more than doubling the performance of the previous version) and Caffe.

4 Conclusions

While undeniably very powerful, the optimal architectures and best hyper-parameters of deep neural networks are also quite hard to set up and select. However, when properly tuned, they can often produce better results than other classical models, as we have demonstrated here on two wind energy problems. The use of weather variables gives to both problems a bi-dimensional input structure; moreover, these variables can be naturally seen as input channels. This may suggest a reason why the best deep results were obtained using convolutional

layers. Deep network training is also very computationally demanding but, on the other hand, lends itself extremely well to the use of GPUs and the large speed-ups that they allow.

In any case, the work presented here has to be considered as a first step. A first line of further work is to consider other convolutional architectures, specially of the AlexNet type ([14]). Another natural option is to try to reduce variance by combining several deep models (notice that they have naturally a low bias). The usual choice in standard MLPs is to repeat training from different random initializations but given the high validation variability during training, a simpler, less costly possibility is to select a certain number M of the models with smallest validation that were obtained in a single training run as the ones followed here.

Furthermore, the tremendous activity in deep learning is producing a large number of proposals for network initialization and architectures as well as model training and regularization. We are also pursuing some of these options.

Acknowledgments. With partial support from Spain's grants TIN2013-42351-P (MINECO) and S2013/ICE-2845 CASI-CAM-CM (Comunidad de Madrid), and the UAM-ADIC Chair for Data Science and Machine Learning.

The second author is also kindly supported by the FPU-MEC grant AP-2012-5163. The authors gratefully acknowledge the use of the facilities of Centro de Computación Científica (CCC) at UAM and thank Red Eléctrica de España for kindly supplying wind energy production data.

References

1. European center for medium-range weather forecasts. <http://www.ecmwf.int/>
2. Global forecast system. <http://www.emc.ncep.noaa.gov/index.php?branch=gfs>
3. Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I.J., Bergeron, A., Bouchard, N., Bengio, Y.: Theano: new features and speed improvements. In: Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop (2012)
4. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems 19 (NIPS 2006), pp. 153–160 (2007). <http://www.iro.umontreal.ca/~lisa/pointeurs/BengioNips2006All.pdf>
5. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. In: Proceedings of the Python for Scientific Computing Conference (SciPy), June 2010. Oral Presentation
6. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology **2**(3), 27:1–27:27 (2011). software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>
7. Ciresan, D.C., Meier, U., Masci, J., Gambardella, L.M., Schmidhuber, J.: Flexible, high performance convolutional neural networks for image classification. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011, Barcelona, Catalonia, Spain, July 16–22, 2011, pp. 1237–1242 (2011). <http://ijcai.org/papers11/Papers/IJCAI11-210.pdf>

8. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, pp. 249–256, May 2010
9. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, April 2011
10. Goodfellow, I.J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F., Bengio, Y.: Pylearn2: a machine learning research library. arXiv preprint [arXiv:1308.4214](http://arxiv.org/abs/1308.4214) (2013). <http://arxiv.org/abs/1308.4214>
11. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR* abs/1502.01852 (2015). <http://arxiv.org/abs/1502.01852>
12. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. *Science* **313**(5786), 504–507 (2006). <http://www.sciencemag.org/content/313/5786/504.abstract>
13. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R.B., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. *CoRR* abs/1408.5093 (2014). <http://arxiv.org/abs/1408.5093>
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc. (2012). <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
15. Kruger, N., Janssen, P., Kalkan, S., Lappe, M., Leonardis, A., Piater, J., Rodriguez-Sanchez, A., Wiskott, L.: Deep hierarchies in the primate visual cortex: What can we learn for computer vision? *IEEE Transactions on Pattern Analysis and Machine Intelligence* **35**(8), 1847–1871 (2013)
16. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**(11), 2278–2324 (1998)
17. LeCun, Y., Bottou, L., Orr, G.B., Müller, K.-R.: Efficient BackProp. In: Orr, G.B., Müller, K.-R. (eds.) *Neural Networks: Tricks of the Trade*. LNCS, vol. 1524, pp. 9–50. Springer, Heidelberg (1998)
18. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* **15**, 1929–1958 (2014). <http://jmlr.org/papers/v15/srivastava14a.html>
19. Sutskever, I., Martens, J., Dahl, G.E., Hinton, G.E.: On the importance of initialization and momentum in deep learning. In: Dasgupta, S., Mcallester, D. (eds.) *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, vol. 28, pp. 1139–1147. *JMLR Workshop and Conference Proceedings*, May 2013. <http://jmlr.org/proceedings/papers/v28/sutskever13.pdf>

Chapter 4

Deep Fisher Discriminant Analysis and Imbalanced Classification

4.1 Summary

Highly imbalanced datasets are common in real world problems, making the topic of imbalanced classification very interesting for practical applications. On the other hand, Fisher Discriminant Analysis (FDA), an elegant dimensionality reduction technique for classification problems, could be a nice tool in such applications, but its linear nature poses a strong limitation on its performance.

This series of publications explores the partial equivalence between the classical eigenanalysis-based solution to FDA and an alternative DNN-based solution that, being iterative and with a computational cost scaling linearly with the sample size, allows to apply FDA to big data problems. As detailed in [87], FDA for two-class problems can be reduced to Least Squares. This result was extended by Zhihua Zhang, Guang Dai, Congfu Xu and Michael I. Jordan in [33, 88] to multiclass problems. Also, the use of MLPs instead of linear models for such feature transformation results in a non-linear extension to FDA comparable in principle to Kernel Fisher Discriminant Analysis (KFDA), without the drawback of the high computational costs associated to kernel-based methods. Finally, the performance of MLP-based models combined with the use of state of the art regularization and activation techniques and proper error and score functions is studied in detail and compared with KFDA.

4.2 Publications

We detail next the publications of this chapter, give their abstracts, and briefly list their main contributions for the sake of convenience.

Main publication

- David Díaz-Vico and José R. Dorronsoro. Deep least squares fisher discriminant analysis. *IEEE Transactions on Neural Networks and Learning Systems*, 31(8):2752–2763, 2020

Abstract: While being one of the first and most elegant tools for dimensionality reduction, Fisher linear discriminant analysis (FLDA) is not currently considered among the top methods for feature extraction or classification. In this paper, we will review two recent approaches to FLDA, namely, least squares Fisher discriminant analysis (LSFDA) and regularized kernel FDA (RKFDA) and propose deep FDA (DFDA), a straightforward nonlinear extension of LSFDA that takes advantage of the recent advances on deep neural networks. We will compare the performance of RKFDA and DFDA on a large number of two-class and multiclass problems, many of them involving class-imbalanced data sets and some having quite large sample sizes; we will use, for this, the areas under the receiver operating characteristics (ROCs) curve of the classifiers considered. As we shall see, the classification performance of both methods is often very similar and particularly good on imbalanced problems, but building DFDA models is considerably much faster than doing so for RKFDA, particularly in problems with quite large sample sizes.

Contributions: This article was published as an substantial extension of [29], where the first idea of a DNN based alternative to FDA was proposed. Besides a more thorough description of the proposed Deep Fisher Discriminant Analysis (DFDA), the main additional contributions are:

- The formulation of Kernel Discriminant Analysis (KDA) in the same terms as DFDA. Efficient algorithms for training and inference of both FDA and KFDA were also proposed.
- The proposal of suitable scoring functions for regularized Kernel Fisher Discriminant Analysis (RKFDA) and deep Fisher Discriminant Analysis (DFDA) based on the computation of receiver operating characteristic (ROC) curves and area under the curve (AUC) values.
- An extensive comparison of RKFDA and DFDA over a large number of two-class and multiclass data sets, many of them involving imbalanced problems and moderately large sample sizes, which show that both give essentially the same classification results but DFDA has a much lower computational cost.

In particular, the experimental methodology was significantly improved by the use of more appropriate metrics and the availability of an extended dataset repository that included balanced and unbalanced, binary and multiclass classification problems, and featured high- and low-dimensional examples, with dataset sizes ranging from around 100 to over 500,000. This resulted in a more solid hypothesis testing, as in [22]. Also, a Scikit-learn-compatible software library with the proposed implementation of KFDA was published as [89].

Other publications

- David Díaz-Vico, Adil Omari, Alberto Torres-Barrán, and José Ramón Dorronsoro. Deep fisher discriminant analysis. In Ignacio Rojas, Gonzalo Joya,

and Andreu Catala, editors, *Advances in Computational Intelligence*, pages 501–512, Cham, 2017. Springer International Publishing

Abstract: Fisher Discriminant Analysis (FDA)’ linear nature and the usual eigen-analysis approach to its solution have limited the application of its underlying elegant idea. In this work we will take advantage of some recent partially equivalent formulations based on standard least squares regression to develop a simple Deep Neural Network (DNN) extension of Fisher’s analysis that greatly improves on its ability to cluster sample projections around their class means while keeping these apart. This is shown by the much better accuracies and g scores of class mean classifiers when applied to the features provided by simple DNN architectures than what can be achieved using Fisher’s linear ones.

Contributions: This congress paper exploits the partial equivalence between the eigenvalue-based analytical solution of FDA and the solution obtained with standard Linear Regression with some suitably transformed targets. This iterative method to calculate a solution of FDA does not involve any eigenanalysis, and its cost scales linearly with the size of the training sample, making it viable for big data problems. Moreover, the use of MLPs instead of simpler Linear Regression adds non-linear capabilities to the resulting feature transformation technique, allowing for a significant improvement on the results for classification problems when paired with an appropriate distance-based classifier.

- David Díaz-Vico, Aníbal R. Figueiras-Vidal, and José R. Dorronsoro. Deep mlps for imbalanced classification. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2018

Abstract: Classification over imbalanced datasets is a highly interesting topic given that many real-world classification problems present a concrete class with a much smaller number of patterns than the others. In this work we shall explore the use of large, fully connected and potentially deep MLPs in such problems. We will consider simple MLPs, with ReLU activations, softmax outputs and categorical cross-entropy loss, showing that, when properly regularized, these relatively straightforward MLP models yield state of the art results in terms of the areas under the ROC curve for both two-class problems (the usual focus in imbalanced classification) as well as for multi-class problems.

Contributions: This congress paper briefly reviewed the state of the art in imbalanced classification problems and explored the use of MLPs with appropriate regularization and activation techniques in such problems. The importance of a wisely chosen metric to optimize during hyper-parameter search was a central topic in the analysis. The experimental results showed that MLPs paired with modern regularization, initialization and activation techniques, using an appropriate metric to evaluate hyper-parameter configurations, can be competitive with the current state of the art. The set of tools used in the experiments included Scipy [90], Scikit-learn [20], Keras [66], Tensorflow [58] and Sacred [91], and served as a check of the maturity and reliability of this Python ML stack to be used in further publications.

Deep Least Squares Fisher Discriminant Analysis

David Díaz-Vico¹ and José R. Dorronsoro

Abstract—While being one of the first and most elegant tools for dimensionality reduction, Fisher linear discriminant analysis (FLDA) is not currently considered among the top methods for feature extraction or classification. In this paper, we will review two recent approaches to FLDA, namely, least squares Fisher discriminant analysis (LSFDA) and regularized kernel FDA (RKFDA) and propose deep FDA (DFDA), a straight-forward nonlinear extension of LSFDA that takes advantage of the recent advances on deep neural networks. We will compare the performance of RKFDA and DFDA on a large number of two-class and multiclass problems, many of them involving class-imbalanced data sets and some having quite large sample sizes; we will use, for this, the areas under the receiver operating characteristics (ROCs) curve of the classifiers considered. As we shall see, the classification performance of both methods is often very similar and particularly good on imbalanced problems, but building DFDA models is considerably much faster than doing so for RKFDA, particularly in problems with quite large sample sizes.

Index Terms—Deep neural networks (DNNs), Fisher discriminant analysis (FDA), kernel discriminant analysis, nonlinear classifiers.

NOMENCLATURE

c	Number of classes.
d	Number of pattern features.
N, N_1, \dots, N_c	Total and class sample sizes.
I, I_d	$d \times d$ identity matrix.
$\mathbf{1}_N$	All ones N -dimensional vector.
X	$N \times d$ sample data matrix.
m	Sample mean.
E	$N \times c$ one hot encoding label matrix.
H	$N \times N$ centering matrix.
Π	$c \times c$ diagonal matrix with $\Pi_{ii} = N_i$.
S_B, S_T	Between-class and total sample covariance matrices.
s_B, s_T	Between-class and total projected covariance matrices.
λ	L_2 regularization parameter.
D_λ	Inverse of $S_T + \lambda I$.
$k(\cdot, \cdot)$	Kernel.

Manuscript received April 27, 2018; revised October 2, 2018, January 25, 2019, and February 21, 2019; accepted March 11, 2019. This work was supported in part by Mineco under Grant TIN2016-76406-P, in part by the Comunidad de Madrid under Grant S2013/ICE-2845 CASI-CAM-CM, in part by the Project FACIL—Ayudas Fundación BBVA a Equipos de Investigación Científica 2016, in part by the Universidad Autónoma de Madrid (UAM)—Asociación para el Desarrollo de la Ingeniería del Conocimiento (ADIC) Chair for Data Science and Machine Learning, and in part by the Instituto de Ingeniería del Conocimiento. (Corresponding author: David Díaz-Vico.)

The authors are with the Dpto. Ing. Informática, Instituto de Ingeniería del Conocimiento, Universidad Autónoma de Madrid, 28049 Madrid, Spain (e-mail: david.diaz.vico@outlook.com).

Digital Object Identifier 10.1109/TNNLS.2019.2906302

K	Kernel matrix.
C	HKH matrix.
\mathcal{W}	Generic weight set of a DNN.
w_0, W	Linear output components of \mathcal{W} .
$\hat{\mathcal{W}}$	Components of \mathcal{W} excluding w_0 and W .
$s_k(x)$	Generic scoring for class k acting on x .
π_i	Prior probability of class i .
$P(0 x), P(1 x)$	Posterior probabilities of classes 0, 1 conditioned on x .
$\text{TPR}(t), \text{FPR}(t)$	True and false positive rates associated with a threshold t .

I. INTRODUCTION

IMBALANCED classification is certainly among the most important problems in machine learning, and as such, it has received a wide attention [1]. This has been particularly the case since the mid-2000s, where articles such as [2] and [3] drew the scientific community to work on a topic that, although recognized as relevant, up to that, the moment had received only scattered attention [4]–[7]. Over time, it has been possible to group [8] the many proposals for handling imbalanced problems into two general approaches, sample-based procedures or algorithm-based ones. Sample-based algorithms usually try to correct the imbalance by undersampling the largest class (see [9]), oversampling the smaller class (as in the celebrated Synthetic Minority Over-Sampling Technique approach [10]), or applying both in combination with an ensemble classifier [11]. Given the computational cost of rebalancing and of working with ensembles, simple classifiers such as decision trees are often used. On the other hand, algorithm-based methods involve stronger classifiers and seek to correct their natural bias toward the larger classes. A natural way to achieve this is to introduce imbalance correcting classification costs [7], or to modify a classifier’s loss function so that the influence of the smaller classes is increased [12]. Neural networks for imbalanced classification were studied under this second approach. In fact, it was early recognized [13] that the usual one-hot target encoding often used in neural classification implicitly and strongly favored the larger class. To correct this, it was proposed in [13] that an alternate coding of class patterns which incorporated class size information into the network’s training.

These observations suggest that classification methods that are implicitly aware of class information should be of interest when seeking efficient classifiers in imbalanced problems. One such method is Fisher linear discriminant analysis (FLDA [14], [15]). FLDA is based on having competing spreads for the within-class covariances (small) and the between-class covariance (large), and it is most often applied to obtain useful data representations. However, while it can

also be used to build distance-based classifiers, it is not frequently applied as such. This is often the case of purely linear classifiers, but another reason is that its goal of keeping class means as far as possible while at the same time having small class variances, can be optimally achieved only when all classes have Gaussian-distributed features with a common covariance matrix.

Nevertheless, FLDA is at the core of several successful methods for important problems. For instance, in face recognition and person reidentification, the well known Fisher face method [16], which initially relied on models relatively close to Fisher’s original proposal, has been progressively improved by adding a nonlinear processing of the images to be identified. Another well-known extension is kernel Fisher discriminant analysis (KFDA [17]), which essentially applies linear FDA over nonlinear extensions of the initial features and takes advantage of the kernel trick to avoid having to explicitly build them. KFDA initially addressed binary classification but has been extended to multiclass problems in [18]. Moreover, the approach in [18], regularized KFDA (RKFDA) puts on a clear footing on how to deal with regularization in KFDA. However, on the other hand, working with kernels requires $N \times N$ matrix computations, with N sample size, which may make it too expensive, if not unfeasible, on large problems. Other kernel extensions of FLDA have been proposed, such as [19] for face recognition, [20] for radial basis function-based classification, or [21] for general feature extraction and recognition.

After the huge success of deep neural networks (DNNs), a clear way for the nonlinear extension of linear methods is to place them at the last hidden layer of a possibly deep multilayer perceptron (MLP). In fact, the recent advances in DNNs have greatly simplified the training of networks with a very large number of layers and hundreds of thousands of weights. To this, we can add the widespread availability of DNN tools such as Theano [22], CNTK [23], Torch [24], MXNET [25], or TensorFlow [26], endowed with compilation procedures that automatically compute backpropagation gradients for cost functions much more general than the cross entropy or square errors that had traditionally been applied in classification or regression. In a slight abuse of language, here, we will use the term “deep” with precisely this meaning of networks defined, initialized, and optimized using these new advanced techniques, rather than implying a large number of hidden layers. In fact, a large number of layers and the correspondingly large number of weights are dependent on having very large training samples with sizes in the millions of patterns. Here, we will consider samples sizes up to the hundreds of thousands and networks with up to five layers; note that while their training is nowadays almost routine, it was simply not possible until a few years ago, as the modern initialization, activation, and optimization techniques were not yet available.

It is thus natural trying to take advantage of this for Fisher analysis, but it is not easy to blend eigencalculations on the covariance matrices of the last hidden layer with the usual backpropagation training of DNNs; see [27] for an early attempt in this direction, see [28] where FLDA is

preceded by nonlinear transformations learned by DNNs in a semisupervised fashion, and see [29] where the direct optimization of FLDA’s eigenvalue-based criterion is proposed. Other examples are [30] for person reidentification or [31] for gender detection. A much simpler way for such an extension is given by the initial results in [32] and [33], and particularly, the proposals in [34] and [18] show how to relate FLDA with a least squares regression (LSR) over properly defined targets (see also [35] for a different least squares approach). We shall make extensive use of this and also of the isometry that is shown in [18] and [34] to exist between the FLDA projections and those induced by the LSR solution. This isometry implies that equivalent FLDA and LSR distance-based classifiers can be defined in terms of k -nearest neighbors, minimum class-mean distances, or as done here, distance-based scores.

As a consequence, a simple way to nonlinearly extend the preceding is to replace linear LSR by a DNN counterpart, where the outputs of a DNN will now approximate appropriate class-based LSR targets. Of course, regularization is also mandatory for DNNs, which involves not only a careful choice of penalties but also an adequate criterion function to be minimized during hyperparametrization. This DNN approach to FDA is our main contribution here (see [36] for a preliminary version), to which we can add the following.

- 1) The proposal of suitable scoring functions for RKFDA and deep FDA (DFDA) based on the computation of receiver operating characteristic (ROC) curves and area under the curve (AUC) values.
- 2) An extensive comparison of RKFDA and DFDA over a large number of two-class and multiclass data sets, many of them involving imbalanced problems, which show that DFDA gives essentially the same classification results but with a much lower computational cost.

This paper is organized as follows. In Section II, we shall briefly review classical FLDA, and in Section III, we give a streamlined exposition of the RKFDA approach in [18]. In Section IV, we will review the equivalence in [18] and [34] between a concrete LSR problem and FLDA. We will propose in Section V a distance-based scoring function for both DFDA and RKFDA that facilitates the computation of AUC values which we will use in order to optimize the regularization parameter of DFDA and RKFDA. In Section VI, we will extensively compare the performance of DFDA and that of RKFDA over a number of substantially imbalanced two-class problems, whereas, in Section VII, we will do so on several large scale problems that are either imbalanced or multiclass or both; we also include, here, results for the largest data sets among those considered in [17]. While, as we shall see, the classification performance of RKFDA and DFDA is essentially the same, the computational costs of DFDA are considerably smaller and make it possible to apply DFDA to large problems for which the large size of the kernel matrix makes the use of RKFDA too costly. Finally, a brief discussion will be given as well as pointers to further work. We also mention that Python code for building RKFDA and DFDA models and performing our experiments are available at a GitHub repository.

II. FISHER'S LINEAR DISCRIMINANT ANALYSIS

A. Generalized Eigenproblem

Recall that FLDA seeks to concentrate its projections around their class means while, at the same time, keeping apart these class means. Several target functions have been proposed, see [14, Sec. 10.2], where many of them are shown to be equivalent. In this paper, we will maximize the trace criterion

$$g(A) = \text{trace}(s_T^{-1}s_B) = \text{trace}((A^T S_T A)^{-1}(A^T S_B A)) \quad (1)$$

where A is the $d \times q$ projection matrix, S_B and S_T are the sample between-class and total covariance matrices, respectively, and s_B and s_T denote the between-class and total covariances of the projections $z = A^T x$, see [18, Sec. 2.2], for more details. Assuming q to be such that s_B has rank q and s_T is invertible, solving $\nabla_A g = 0$ leads to $0 = -2 S_T A s_T^{-1} s_B s_T^{-1} + 2 S_B A s_T^{-1}$, i.e., to the problem of finding A such that $S_B A = S_T A s_T^{-1} s_B$.

Given that, for any M and an invertible Q , $\text{trace}(Q^{-1} M Q) = \text{trace}(Q Q^{-1} M) = \text{trace}(M)$, the solution of (1) is unique modulo any such a $q \times q$ transformation Q of the A projections, which will not change the cost function $g(A)$. In particular, if $s_T^{-1} s_B = U \Gamma U^T$ is the SVD of $s_T^{-1} s_B$, we can replace the previous problem with the following equivalent eigenproblem:

$$S_B A = S_T A \Gamma \quad (2)$$

with Γ the eigenvalues of $s_T^{-1} s_B$ and, hence, the nonzero eigenvalues of $S_T^{-1} S_B$. If S_T is invertible, solving (2) is equivalent to solving $S_T^{-1} S_B A = A \Gamma$, we then have

$$g(A) = \text{trace}(s_T^{-1} s_B) = \text{trace } \Gamma = \gamma_1 + \dots + \gamma_q. \quad (3)$$

We can maximize this simply by selecting the q largest eigenvalues in Γ after sorting them in descending order, together with some convenient normalization of their associated eigenvectors. In fact, note that a normalization has to be introduced given that the maximizer of (1) is not uniquely defined; the usual choice is to have $A^T S_T A = I_q$ but, here, we will consider solutions B such that $B^T S_T B = \Gamma$. Observe that we can move from A to B simply by setting $B = A \Gamma^{1/2}$ and vice versa.

Finally, it may be the case that S_T does not have full rank. While often S_T^{-1} is then replaced with the Moore–Penrose inverse of S_T , here, we will consider regularized discriminant analysis [37], working with the positive definite matrix $S_T + \lambda I$ for some $\lambda > 0$ and solving then the eigenvalue problem

$$(S_T + \lambda I)^{-1} S_B A = A \Gamma. \quad (4)$$

B. Solving the Generalized Eigenproblem

At first sight, solving (4) would require the computation of:

- 1) S_B and $D_\lambda = (S_T + \lambda I)^{-1}$ at a cost $O(Nd^2) + O(d^3)$;
- 2) $D_\lambda S_B$ at a cost $O(d^3)$;
- 3) A and Γ from the SVD of $D_\lambda S_B$ at a cost $O(d^3)$.

Thus, besides computing S_T , three steps with a cost $O(d^3)$ are involved. To improve on this, first, observe that if X denotes the $N \times d$ data matrix, we have

$$S_T = X^T H X, \quad S_B = X^T H E \Pi^{-1} E^T H X \quad (5)$$

Algorithm 1 Solving Fisher Linear Discriminant Method

- 1 Read $X, \lambda, E, (N_1, \dots, N_c)$
- 2 Compute $\Pi = \text{diag}(N_1, \dots, N_c)$
- 3 Compute $S_T = X^T H X$
- 4 Compute $D_\lambda = (S_T + \lambda I_d)^{-1}$
- 5 Compute $Q = X^T H E \Pi^{-1/2}$ and $R = Q^T D_\lambda Q$
- 6 Compute the SVD $R = V \Gamma V^T$
- 7 Return Γ, V, D_λ

where the superscript t indicates the transpose, Π denotes the $c \times c$ diagonal matrix with $\Pi_{ii} = N_i$, the number of sample patterns in class i , E denotes the $N \times c$ one-hot encoding matrix, and H is the centering matrix

$$H = I_N - \frac{1}{N} \mathbf{1}_N \mathbf{1}_N^T \quad (6)$$

with $\mathbf{1}_N$ the all ones N -dimensional vector; in particular, the p th row of the $N \times d$ matrix $H X$ equals $x_p^t - m^t$, with x_p the p th sample vector. Moreover

$$S_B = X^T H E \Pi^{-1/2} \Pi^{-1/2} E^T H X = Q Q^T \quad (7)$$

where we write $Q = X^T H E \Pi^{-1/2}$. Defining, now, the $c \times c$ matrix

$$R = Q^T D_\lambda Q \quad (8)$$

if $R = V \Gamma V^T$ is its reduced SVD decomposition (i.e., we only consider the $q \leq c$ nonzero eigenvalues in Γ), then it can be easily seen that the pair $(A = D_\lambda Q V, \Gamma)$ verifies $D_\lambda S_B A = A \Gamma$, i.e., it solves (4), for we have

$$\begin{aligned} D_\lambda S_B A &= D_\lambda Q Q^T D_\lambda Q V = D_\lambda Q R V \\ &= D_\lambda Q V \Gamma = A \Gamma. \end{aligned}$$

This suggests to solve (4) through the steps in Algorithm 1, whose computational cost is:

- 1) $O(Nd^2)$ to compute S_T at step 3 of the algorithm;
- 2) $O(d^3)$ to compute D_λ in step 4;
- 3) $O(dNc) + O(dc^2)$ to compute Q and $O(cd^2 + c^2 d)$ to compute R in step 5;
- 4) $O(c^3)$ to compute V, Γ from the SVD of R in step 6;
- 5) $O(dq^2)$ to compute $A = D_\lambda Q V$ after the return in step 7.

There is thus only one step with a cost $O(d^3)$ and when $d > N > c$, this should improve on the first approach.

III. REGULARIZED KERNEL FISHER DISCRIMINANT ANALYSIS

A. Plain Regularized KFDA Problem

In what follows we loosely follow the discussion in [18]. We also point out that, as mentioned in [18], the treatment below is slightly different from that in [17]. We assume starting patterns $x \in \mathbf{R}^d$ and their expansions $\tilde{x} = \Phi(x) \in \mathbf{R}^D$ in some (quite) large dimensional space (i.e., $D \gg d$). We will denote the

expanded data matrix as \tilde{X} and we can define the between and total covariance matrices of the expansions as

$$\tilde{S}_B = \tilde{X}^t H E \Pi^{-1} E^t H \tilde{X}, \quad \tilde{S}_T = \tilde{X}^t H \tilde{X} \quad (9)$$

with H again the centering matrix $H = I_N - (1/N)\mathbf{1}_N\mathbf{1}_N^t$ and E the one-hot encoding matrix. The generalized regularized eigenvalue problem now to be solved is

$$\tilde{S}_B \tilde{A} = (\tilde{S}_T + \lambda I_D) \tilde{A} \tilde{\Gamma} \quad (10)$$

with $\tilde{A} \in \mathbf{R}^{D \times q}$. This is [18, eq. (14)], shown also in that paper to be equivalent to its kernel version problem in [18, eq. (16)].

Given that now $D \gg c$ is certain, the way to proceed according to the previous discussion for FLDA would be to set $\tilde{Q} = \tilde{X}^t H E \Pi^{-1/2}$ and to define again the $c \times c$ matrix R as

$$R = \tilde{Q}^t \tilde{D}_\lambda \tilde{Q} = \Pi^{-1/2} E^t H \tilde{X} \tilde{D}_\lambda \tilde{X}^t H E \Pi^{-1/2} \quad (11)$$

where $\tilde{D}_\lambda = (\tilde{S}_T + \lambda I)^{-1}$ and R has rank $q \leq c - 1$; we then get its reduced SVD decomposition $R = V \Gamma V^t$ and somehow compute the projection matrix $\tilde{A} = \tilde{D}_\lambda \tilde{Q} V$. However, this is not feasible as all the preceding computations would have to be performed on the expanded \tilde{X} , something we want to avoid.

To do so, we will rewrite $\tilde{D}_\lambda \tilde{X}^t H$ as follows. Set $C = H \tilde{X} \tilde{X}^t H = H K H$, where we will call $K = \tilde{X} \tilde{X}^t$ the *kernel* matrix as we assume its entries $\tilde{x}_p \cdot \tilde{x}_q$ to be computed on the initial features x through a suitable kernel $k(x_p, x_q)$, and set also $\Delta_\lambda = (C + \lambda I_N)^{-1}$. Then, we have

$$\begin{aligned} \tilde{X}^t H &= \tilde{X}^t H (C + \lambda I_N) \Delta_\lambda \\ &= (\tilde{X}^t H H \tilde{X} \tilde{X}^t H + \lambda \tilde{X}^t H) \Delta_\lambda \\ &= (\tilde{X}^t H H \tilde{X} + \lambda I_D) \tilde{X}^t H \Delta_\lambda \\ &= \tilde{D}_\lambda^{-1} \tilde{X}^t H \Delta_\lambda \end{aligned}$$

therefore, it follows that

$$\tilde{D}_\lambda \tilde{X}^t H = \tilde{X}^t H \Delta_\lambda \quad (12)$$

which, in turn, implies

$$\begin{aligned} R &= \Pi^{-1/2} E^t H \tilde{X} (\tilde{D}_\lambda \tilde{X}^t H E) \Pi^{-1/2} \\ &= \Pi^{-1/2} E^t H \tilde{X} (\tilde{X}^t H \Delta_\lambda) E \Pi^{-1/2} \\ &= \Pi^{-1/2} E^t (H \tilde{X} \tilde{X}^t H) \Delta_\lambda E \Pi^{-1/2} \\ &= \Pi^{-1/2} E^t C \Delta_\lambda E \Pi^{-1/2} \end{aligned} \quad (13)$$

where we use parentheses for easier reading.

Therefore, the kernel matrix is just what we need to obtain the matrix R without having to handle the extended data matrix \tilde{X} and to compute R 's SVD to get V and Γ . Using (12) again, the resulting projecting matrix \tilde{A} would then be

$$\begin{aligned} \tilde{A} &= \tilde{D}_\lambda \tilde{Q} V = \tilde{D}_\lambda \tilde{X}^t H E \Pi^{-1/2} V \\ &= \tilde{X}^t H \Delta_\lambda E \Pi^{-1/2} V. \end{aligned} \quad (14)$$

At first sight, \tilde{A} seems to require \tilde{X} but, again, we can avoid this when computing the Fisher projections. In fact, the projection of the expansion \tilde{x} of a new x would be

$$\begin{aligned} z &= \tilde{A}^t (\tilde{x} - \tilde{m}) = V^t \Pi^{-1/2} E^t \Delta_\lambda H \tilde{X}^t (\tilde{x} - \tilde{m}) \\ &= V^t \Pi^{-1/2} E^t \Delta_\lambda H \left(k_x - \frac{1}{N} K \mathbf{1}_N \right) \end{aligned} \quad (15)$$

Algorithm 2 Training Phase of the Regularized Kernel Fisher Discriminant Method

- 1 Read $K, \lambda, E, (N_1, \dots, N_c)$
 - 2 Compute $\Pi = \text{diag}(N_1, \dots, N_c)$
 - 3 Compute $C = H K H$
 - 4 Compute $\Delta_\lambda = (C + \lambda I_N)^{-1}$
 - 5 Compute $F = E^t C$ and $G = \Delta_\lambda E$
 - 6 Compute $P = \Pi^{-1/2} F, Q = G \Pi^{-1/2}$ and $R = P Q$
 - 7 Compute the condensed SVD $R = V \Gamma V^t$
 - 8 Return $\Gamma, V, \Delta_\lambda$
-

where k_x is the vector $(k(x, x_1), \dots, k(x, x_N))^t$ and which only involves kernel operations. Note that when $k(x, x') = x \cdot x'$ (i.e., $\tilde{x} = x$), we simply recover the previous solution of FLDA.

We next make explicit the corresponding training and testing algorithms for this RKFDA procedure.

B. Train and Test Algorithms for the RKFDA Problem

We describe, here, [18, Algorithm 5] in a more detailed form, handling separately what would be its training phase (Algorithm 2) and its testing phase (Algorithm 3) and with an eye to the detailed computational analysis we given in the following. In the training phase, we simply compute R in (13) and perform a SVD on it, getting V and Γ such that

$$R = V \Gamma V^t \quad (16)$$

where Γ contains the $q \leq c - 1$ nonzero eigenvalues of R and V is made of orthogonal eigenvectors.

These steps to obtain V and Γ are put together in Algorithm 2 for the training phase of RKFDA, whose computational costs run as follows.

- 1) Since we have $C = H K H = K - (1/N)(D^K + (D^K)^t) + (1/N)V^K$, with:
 - a) D^K the $N \times N$ matrix $D^K = (d^K, \dots, d^K)$ with d^K the degree vector $d^K = (d_1^K, \dots, d_N^K)^t$ and $d_p^K = \sum_q K_{p,q}$
 - b) $V_{pq}^K = v^K = \sum_{p,q} K_{p,q}$
 computing C in line 3 does not essentially involve float operations once the kernel matrix K is available; computing it has a cost of $O(N^2\kappa)$, where κ is the cost of a kernel computation $k(x, x')$.
- 2) Computing $\Delta_\lambda = (C + \lambda I_N)^{-1}$ in line 4 has a $O(N^3)$ cost.
- 3) Computing $F = E^t C$ and $G = \Delta_\lambda E$ in line 5 has a cost of $O(cN^2)$.
- 4) Computing $P = \Pi^{-1/2} F, Q = G \Pi^{-1/2}$ and $R = P Q$ in line 6 has a cost of $O(c^2 N)$.
- 5) Computing the condensed SVD $R = V \Gamma V^t$ in line 7 has also a cost of $O(c^3)$.

Since we may expect $N \gg c$, the most expensive operation is computing $\Delta_\lambda = (C + \lambda I_N)^{-1}$.

With regards to the test phase, where we apply the projections $A^t \tilde{x}$ to the expansions \tilde{x} of new, unseen x , the required steps are outlined in Algorithm 3 for getting these projections

Algorithm 3 Test Phase of the Regularized Kernel Fisher Discriminant Method

- 1 Read $k(\cdot, \cdot)$, $S = \{x_1, \dots, x_n\}$, $\Delta_\lambda = (C + \lambda I_N)^{-1}$, E , (N_1, \dots, N_c) , Γ_q , V_q
 - 2 Read test sample $S_{test} = \{x'_1, \dots, x'_M\}$
 - 3 Compute the test kernel matrix $K_S = (k'_1, \dots, k'_M)$, with

$$k'_p = (k(x'_p, x_1), \dots, k(x'_p, x_N))^t - \bar{k}'_p \mathbf{1}_N \quad (17)$$
 and $\bar{k}'_p = \frac{1}{N} \sum_q k(x'_p, x_q)$
 - 4 Compute $C_S = HK_S$
 - 5 Compute $\Pi = \text{diag}(N_1, \dots, N_c)$
 - 6 Compute $T = V^t \Pi^{-1/2} E^t \Delta_\lambda$
 - 7 Return TC_S
-

on a test sample $S_{test} = \{x'_1, \dots, x'_M\}$ with M new patterns. Its main computational costs are those of the following.

- 1) Computing K_S in line 3 with a cost of $O(NM\kappa)$, where we recall that κ denotes the cost of a kernel computation $k(x, x')$.
- 2) Computing $C_S = HK_S = K_S - (1/N)\mathbf{1}_N \mathbf{1}'_N K_S$ in line 4 only involves sums but no further kernel operations.
- 3) Computing $T = (V^t \Pi^{-1/2})(E^t \Delta_\lambda)$ in line 6 with a $O(cN^2)$ cost.
- 4) Computing TC_S in line 7 with a $O(cNM)$ cost.

Thus, assuming $N \geq M \geq c$, the overall test cost is dominated by the $O(cN^2)$ of line 6.

Finally, we observe that we have relied on the assumption of a finite dimension D for the \tilde{x} for the sake of motivating and deriving the above-mentioned algorithms. However, we ultimately only need a kernel $k(x, x')$ to compute the matrix R and the projections z in Algorithms 2 and 3. This makes possible to work, for instance, with projections $\Phi(x)$ in a countably infinitely dimensional Hilbert space when Gaussian kernels are used.

IV. DEEP FISHER DISCRIMINANT ANALYSIS

A. Least Squares Regression and FLDA

Let X be the $n \times d$ data matrix, $\mathbf{1}_n$ the all ones vector, and in a two-class problem, let y be the target vector defined by setting $y_p = n/n_1$ for patterns in class 1 and $y_p = -n/n_2$ for those in class 2. Then, it is well known [15] that solving

$$\min_{w_0, w} \frac{1}{2} \|y - \mathbf{1}_n w_0 - Xw\|^2 \quad (18)$$

gives a solution to FLDA. In fact, if m_1 and m_2 are the class means on the original features, then $S_B = (m_1 - m_2)(m_1 - m_2)^t$, and setting $w = S_T^{-1}(m_1 - m_2)$, it is easy to check that $S_T^{-1} S_B w = w\gamma$, with $\gamma = (m_1 - m_2)^t S_T^{-1} (m_1 - m_2)$. In other words, w solves the eigenproblem (4) and, therefore, coincides with a dilation of an FLDA's projection vector.

Many attempts have been made to extend this simple equivalence to multiclass problems. This has been achieved by the essentially equivalent proposals of Ye [34] and, particularly, Zhang *et al.* [18] (see also the proposals by Park and Park [33]). We describe it next.

With $\mathbf{1}_n$ and X as before, let now W be a $d \times q$ matrix, w_0 a $q \times 1$ vector and Y a target matrix to be chosen, and consider the LSR problem of minimizing

$$\min_{w_0, W} \frac{1}{2} \|Y - \mathbf{1}_n w_0 - XW\|^2. \quad (19)$$

Assuming for simplicity, a regular S_T (or working with $S_T + \lambda I$ for some $\lambda > 0$ if not), the optimal W^* solving (19) is

$$W^* = S_T^{-1} X^t H Y \quad (20)$$

with H again the centering matrix. As in Section II, we have here $S_T = X^t H X$ and also

$$S_B = X^t H E \Pi^{-1/2} \Pi^{-1/2} E^t H X = Q Q^t$$

with E the $N \times c$ one-hot encoding matrix. Choosing as targets in (19) the $N \times c$ matrix $Y = H E \Pi^{-1/2}$, the LSR matrix W^* solution is given by

$$W^* = S_T^{-1} X^t H Y = S_T^{-1} X^t H E \Pi^{-1/2} = S_T^{-1} Q. \quad (21)$$

We see now that we can recover from W^* a solution of (4). To do so, consider again the reduced SVD decomposition $R = V \Gamma V^t$ of the $c \times c$ matrix $R = Q^t S_T^{-1} Q$. Setting $B = W^* V$, we have

$$\begin{aligned} S_T^{-1} S_B B &= S_T^{-1} S_B W^* V \\ &= S_T^{-1} Q Q^t W^* V = S_T^{-1} Q Q^t S_T^{-1} Q V \\ &= S_T^{-1} Q R V = S_T^{-1} Q V \Gamma \\ &= W^* V \Gamma = B \Gamma. \end{aligned}$$

Moreover

$$\begin{aligned} B^t S_T B &= V^t (W^*)^t S_T W^* V = V^t Q^t S_T^{-1} S_T S_T^{-1} Q V \\ &= V^t Q^t S_T^{-1} Q V = V^t R V \\ &= \Gamma. \end{aligned}$$

In other words, (B, Γ) is a solution of (4) with the normalization $B^t S_T B = \Gamma$.

However, we would like to avoid working with B as it would require the SVD of R and, instead, derive a suitable projection from the LSR solution W . To achieve this, let us denote by $z = (W^*)^t x$ and $\omega = B^t x$ the W^* and B projection, respectively, of a pattern x . We then have [18]

$$\begin{aligned} \|\omega - \omega'\|_2^2 &= (x - x')^t B B^t (x - x') \\ &= (x - x')^t W^* V V^t (W^*)^t (x - x') \\ &= \|z - z'\|_2^2 \end{aligned}$$

which implies that

$$\|z - \bar{z}_k\|_2 = \|\omega - \bar{\omega}_k\|_2, \quad (22)$$

where $\bar{z}_k, \bar{\omega}_k$ denote the k th class means for the z and ω projections.

Thus, any distance classifier or, more generally, any score based on distances to class means will give the same results when computed either over the least squares z projection or over the linear FDA ones ω . In other words, the LSR solution can be used to define distance-based scores equivalent to the ones which could be defined using Fisher's projection;

we will discuss them in Section V below. This LSR procedure opens the way to a nonlinear, DNN-based approach to Fisher's analysis which we describe in Section IV-B. Before doing so, we point out that it can be easily seen that the row in the target matrix $Y = HE\Pi^{-1/2}$ for a pattern x_p of class k is given by

$$Y_{pk} = \frac{n - n_k}{n\sqrt{n_k}} \quad (23)$$

and $Y_{pk'} = -(\sqrt{n_k}/n)$ for the other components $k' \neq k$. (In [34], these Y values are just multiplied by \sqrt{n} .)

B. Deep Neural Fisher Discriminant Networks

As just argued, a distance-based classifier for a c class problem equivalent to the one resulting from Fisher projections can be obtained through the following steps.

- 1) For a given training matrix X_{tr} , class indicator matrix E_{tr} and targets $Y_{tr} = HE_{tr}\Pi^{-1/2}$, obtain the c -dimensional vector w_0^* and $d \times c$ matrix W^* which solve the LSR problem

$$\min_{w_0, W} \frac{1}{2} \|Y_{tr} - \mathbf{1}_n w_0^t - X_{tr} W\|^2. \quad (24)$$

- 2) Use them to compute the projections $y = w_0^* + (W^*)^t x$ for $x \in D_{tr}$ and their class means $\bar{y}_k = w_0^* + (W^*)^t \bar{x}_k$.
- 3) Assign a test pattern x to a class according to scores defined in terms of the distances between the projection $y = w_0^* + (W^*)^t x$ and the class means \bar{y}_k .

Now, it is natural to define a nonlinear extension by applying the previous LSR steps to nonlinear extensions $z = \Phi(x)$ of the original features x . A simple way is to apply the previous linear steps to the z features on the last hidden layer of a DNN; more precisely, we

- 1) Solve the LSR problem

$$\min_{\mathcal{W}} \frac{1}{2} \|Y_{tr} - f(X_{tr}, \mathcal{W})\|^2 \quad (25)$$

to get an optimal DNN weight set \mathcal{W}^* ; here, X_{tr} is the training matrix, Y_{tr} is the training target matrix defined previously and the matrix $f(X_{tr}, \mathcal{W})$ has rows of the form $f(X_{tr}, \mathcal{W})_p = f(x_p, \mathcal{W})$, with $f(x, \mathcal{W})$ the linear outputs of a deep network with weights \mathcal{W} .

- 2) Compute the DNN projections $y_p = f(x_p, \mathcal{W}^*)$ over X_{tr} and their class means \bar{y}_c .
- 3) Compute for the rows x in a test matrix X_{ts} their DNN projections $y = f(x, \mathcal{W}^*)$ and corresponding scores according to their distances to the \bar{y}_c means and assign them to the class with the highest score.

Let us write the optimal weight as $\mathcal{W}^* = (w_0^*, W^*, \tilde{\mathcal{W}}^*)$, where w_0^*, W^* are the linear weight vector and matrix that connect the last hidden layer with the network outputs; let us also denote as z the last hidden layer features $z = \Phi(x, \tilde{\mathcal{W}}^*)$, with Φ the partial DNN transformation that computes them. Then, the previous w_0^*, W^* also solve the LSR problem (18) over the z features: if not and there were better choices, say \hat{w}_0 and \hat{W} for w_0 and W with a smaller square error over the z features, the DNN weight set $(\hat{w}_0, \hat{W}, \tilde{\mathcal{W}}^*)$ would yield a smaller error than the one defined by the previously

optimal \mathcal{W}^* . As a consequence, any score-based classifier built on the full DNN projections is equivalent to the same score classifier acting over the FLDA projections of the last hidden layer patterns.

We will call this DFDA, as it effectively applies Fisher's standard linear discriminant analysis over the features z learned by training a deep neural model. As mentioned in Section I, we point out that our use of the term "deep network" has more to do with the underlying network architectures and initialization and training techniques that we will use, than with the networks having a large number of hidden layers (which will be at most five in our experiments).

To avoid singularities, we can simply add a regularization term. In its simplest form, we would solve

$$\min_{w_0, W, \tilde{\mathcal{W}}} \frac{1}{2} \|Y - f(X, w_0, W, \tilde{\mathcal{W}})\|^2 + \frac{\lambda}{2} \text{trace}(W^t W + \tilde{W}^t \tilde{W}) \quad (26)$$

where \tilde{W} are the components of $\tilde{\mathcal{W}}$ when layer biases are removed. We shall use this cost function in our experiments. Note that other regularization procedure, such as dropout, could be used for the \tilde{W} weights; on the other hand, $(\lambda/2)\text{trace}(W^t W)$ should be the regularizer of the linear output weights W .

V. SCORING FUNCTIONS AND AUC COMPUTATION

A. Scoring Functions for RKFDA and DFDA

Since we intend to use the AUC as our merit function for model hyperparametrization and test set evaluation, we will transform the RKFDA and DFDA outputs into vector scores with components in a $[0, 1]$ range, where higher values of the score components should reflect outputs closer to a class centroid. The desired $[0, 1]$ score range is reminiscent of that of posterior Probabilities, and in principle, one way of obtaining such scores could be to try to exploit the fact that Fisher's LDA maximizes the posterior class probabilities assuming all sample class densities are given by Gaussians with different means but the same covariance. However, this is most likely not being true of the original features, cannot be checked on the implicit features of RKFDA, and is not guaranteed at all for the deep features at the last hidden layer of a DFDA network. Because of this, we prefer to follow the simple heuristic we describe next.

Note that, for the RKFDA z projections defined by (15), class centroids \bar{z}_k are easily computed and so are the distances to them of new pattern projections. Moreover, for FDA, we have just observed in (22) that the least squares projections have the same centroid distances than those of a Fisher projection. This obviously extends to the outputs of a DFDA network, as they are the least squares projections of the last hidden layer representations.

Thus, for both RKFDA and DFDA, we can easily compute the class-means distances $\|z - \bar{z}_k\|$ of their projections z , distances that we will transform into $[0, 1]$ scores better suited to our subsequent work. To do so, given a validation or test Fisher projection z , we first compute its distances $d_k = d_k(z)$

to the centroids \bar{z}_k , $1 \leq k \leq c$, of the training class projections and then we define the k th score of z as the following normalization of the inverses $1/d_k$, namely,

$$\begin{aligned} s_k = s_k(z) &= \frac{\frac{1}{d_k}}{\sum_j \frac{1}{d_j}} = \frac{1}{\sum_j \frac{d_k}{d_j}} \\ &= \frac{1}{1 + \sum_{j \neq k} \frac{d_k}{d_j}}. \end{aligned}$$

Of course, we have to watch out for zero or near-zero values of d_k . Assuming different class centroids, $d_j = 0$ can only happen at one k ; in such a case, the previous expression would clearly give $s_k = 1$ while we simply take $s_j = 0$ for other j values as they would involve a fraction $1/d_k = \infty$ in the denominator.

B. AUC for Two-Class and Multiclass Problems

Considering first two-class problems, let $s(x)$, $0 \leq s(x) \leq 1$ be a scoring function which gives higher values to positive patterns, i.e., those in class 1; an example could be any estimate $\hat{P}(1|x)$ of the conditional probability of a pattern being from class 1 given its features x . Let f_0, f_1 be the densities of the class conditioned random scores $S_0(x) = s(x|0)$, $S_1(x) = s(x|1)$ which we assume to be independent. Then, for any threshold t , let $c_t(x)$ be the classifier such that $c_t(x) = 1$ iff $s(x) > t$ and let true positive ratios (TPR(t)), false positive ratios (FPR(t)) be its true and false positive ratios, that is,

$$\begin{aligned} \text{TPR}(t) &= P(\{s(x) > t | x \in C_1\}) = \int_t^1 f_1(s) ds \\ \text{FPR}(t) &= P(\{s(x) > t | x \in C_0\}) = \int_t^1 f_0(s) ds. \end{aligned}$$

Note that TPR is equivalent to recall or sensitivity and $1 - \text{FPR}$ is equal to specificity. The ROC curve is defined by the points (FPR(t), TPR(t)) and the area below the ROC is called the AUC. We define AUC in terms of the 1 class but the same value is obtained if defined in terms of the 0 class.

The AUC captures in a single number of the performance of the underlying classifier across all thresholds $0 \leq t \leq 1$. Moreover, since TPR(t) and FPR(t) are computed on the rows and columns on the confusion matrix, the AUC should be more robust on imbalanced problems, as it uses no prior probability information. Finally, it can be shown [38] that

$$\text{AUC} = P(\{s(x) > s(x') : x \in C_1, x' \in C_0\})$$

i.e., the AUC measures the probability that the score of a random positive pattern is larger than that of negative one. In particular, this supports the intuition of a given classifier being preferable to another with smaller AUC.

Contrary to the two-class situation, no clear cut extension of the AUC to a multiclass setting has been given. Conceptually, the volume under the surface (VUS) in [39] is possibly closest to the previous two-class formulation, but it may be quite difficult to compute, especially for more than three classes. Simpler approaches can be derived by computing

and combining several two-class AUC values, such as the *total* AUC [40]

$$\text{AUC}_{\text{total}} = \sum_{i=1}^c \pi_i \text{AUC}_i \quad (27)$$

where, for each i , π_i is the class prior and AUC_i is computed as in a two-class problem with C_i as the positive class; the *macroaverage* AUCs [41], either on its arithmetic

$$\text{AUC}_{\text{macro}} = \frac{1}{c} \sum_i \text{AUC}_i \quad (28)$$

or geometric

$$\text{AUC}_{\text{geom}} = \left(\prod_1^c \text{AUC}_i \right)^{\frac{1}{c}} \quad (29)$$

mean variants; the *microaverage* AUCs of [42, Ch. 13], or finally, the *M-AUC* [43]

$$M = \frac{2}{c(c-1)} \sum_{i < j} \text{AUC}(i, j) \quad (30)$$

which combines $c(c-1)$ two class AUC(i, j). See [44] for more details on the AUC and [38], [41], and [45] for examples of its use in multiclass problems. As just mentioned, there does not seem to be a general agreement on which AUC variant should be used in multiclass problems. Given its simplicity and relative robustness on imbalanced data sets, we shall use geometric macro-AUC in our multiclass experiments.

VI. DFDA VERSUS RKFDA ON IMBALANCED TWO-CLASS PROBLEMS

RKFDA and DFDA over 30 data sets taken from the Keel repository [46]. They all derive from an original set of seven problems (some of them multiclass) whose samples are grouped in various ways to produce 30 different two-class problems with imbalance ratios that range from a minimum of 9.22 (when the classes 0 and 4 of the `glass` problem are to be classified against the class 5) to a maximum of 129.44 (when class 19 of the `abalone` problem is pitted against all others). Their sample sizes, dimensions, and imbalance ratios are given in Table I where the data set names to follow the Keel naming conventions. For all problems considered, we have used the five train-test folds provided in the Keel repository.

We will compare the performance of RKFDA models against DFDA models with a feedforward architecture. The quite popular ReLU function is our choice for the hidden layer activations; as usual in a regression setting, we have linear outputs, and we use Adam over minibatches as the backpropagation optimizer.

In principle, these choices could imply a substantial hyperparameterization cost but we will simplify this as follows. First, we will use Adam's default values for the initial learning rate (0.001) and its β_{t1} (0.9) and β_{t2} (0.999) parameters, as they are quite reliable and robust; similarly, we leave minibatch size at its `scikit-learn` default (200). The second source of hyperparameters could be the number of hidden layers and of units on each of them in the DFDA

TABLE I
DIMENSIONS, SAMPLE SIZES, AND IMBALANCE RATIOS FOR
THE TWO-CLASS PROBLEMS

Dataset	dimension	n. patterns	imbalance ratio
abalone9-18	8	731	16.4
abalone19	8	4174	129.44
ecoli4	7	336	15.8
ecoli-0-1-3-7_vs_2-6	7	281	39.14
ecoli-0-1-4-6_vs_5	6	280	13
ecoli-0-1-4-7_vs_2-3-5-6	7	336	10.59
ecoli-0-1-4-7_vs_5-6	6	332	12.28
ecoli-0-1_vs_5	6	240	11
ecoli-0-3-4-6_vs_5	7	205	9.25
ecoli-0-3-4-7_vs_5-6	7	257	9.28
ecoli-0-6-7_vs_5	6	220	10
glass2	9	214	11.59
glass4	9	214	15.47
glass5	9	214	22.78
glass-0-1-4-6_vs_2	9	205	11.06
glass-0-1-6_vs_2	9	192	10.29
glass-0-1-6_vs_5	9	184	19.44
glass-0-4_vs_5	9	92	9.22
glass-0-6_vs_5	9	108	11
page-blocks-1-3_vs_4	10	472	15.86
shuttle-c0-vs-c4	9	1829	13.87
shuttle-c2-vs-c4	9	129	20.5
vowel0	13	988	9.98
yeast4	8	1484	28.1
yeast5	8	1484	32.73
yeast6	8	1484	41.4
yeast-0-5-6-7-9_vs_4	8	528	9.35
yeast-1-2-8-9_vs_7	8	947	30.57
yeast-1-4-5-8_vs_7	8	693	22.1
yeast-2_vs_8	8	482	23.1

models but here we will simply report our results for networks with a number of hidden layers ranging from 0 to 5, and 100 units each.

This leaves us with the L_2 (or Tikhonov) regularization penalty λ as our only DFDA hyperparameter, for which we explore 50 values evenly spaced on a log scale in the interval $[2^{-30}, 2^{10}]$ selecting the optimal one by k -fold cross validation as described in the following. Prior to this, the DFDA features have been normalized to 0 mean and 1 standard deviation.

The RKFDA models require two hyperparameters, the L_2 regularization penalty λ and the width γ of the Gaussian kernels $\exp(-\gamma \|x - x'\|^2)$; both will be also selected here by cross validation. As before, for λ , we will explore 50 values in the interval $[2^{-30}, 2^{10}]$. In order to select γ , we scale featurewise the RKFDA inputs to a $[0, 1]$ range; note that after this we will have $\|x - x'\|^2 \leq d$, with d pattern dimension. Because of this, we will explore γ values of the form $(2^k/d)$, with k in the $[-10, 10]$ range; in other words, the considered kernels will essentially be powers of a basic exponential e^{-z^2} .

As just mentioned, for each of the five train partitions provided for each data set, optimal λ regularization parameters, and in the RKFDA case, γ values have been obtained by fivefold stratified cross validation using the AUC of the positive class as the scoring function (recall that, as mentioned, $AUC_1 = AUC_0$). Once the optimal λ and, for RKFDA, γ are chosen, we have used them to train individual DFDA and RKFDA models on the train partition and applied them on the test partition, computing afterward the test AUC scores. In Table III, we show, for each problem, the average AUC values over five test splits of the RKFDA and all the DFDA

TABLE II
DIMENSIONS, TRAIN SAMPLE SIZES, NUMBER OF CLASSES, AND
IMBALANCE RATIOS FOR THE LARGE SIZE DATA SETS

Problem	n. patterns	dimension	n. classes	imbalance ratios
a4a	4,781	123	2	3.024
a8a	22,696	123	2	3.122
dna	2,000	180	3	2.265
ijcnn1	49,990	22	2	9.301
letter	15,000	16	26	1.128
pendigits	7,494	16	10	1.085
satimage	4,435	36	6	2.583
shuttle	43,500	9	7	5,684.667
usps	7,291	256	10	2.203
w7a	24,692	300	2	32.368
w8a	49,749	300	2	32.637
<hr/>				
banana	5,300	2	2	1.231
german	1,000	20	2	2.333
image	2,086	18	2	1.323
ringnorm	7,400	20	2	1.020
splice	2,991	60	2	1.225
twonorm	7,400	20	2	1.002
waveform	5,000	21	2	2.036
<hr/>				
combined	78,823	100	3	2.156
covtype.binary	581,012	54	2	1.051
skin_nonskin	245,057	3	2	3.818

classifiers except the one with a single hidden layer that we omit for space and formatting reasons. For easier reading of these values, the table also gives at the bottom the average of the rankings of the six models considered for each problem.

As it can be seen, these average rankings are quite similar for all methods except, as it was to expected, the DFDA_0 model which, being linear, it is in fact equivalent to a standard linear Fisher model (note that, nevertheless, it gives the best test average AUC in some problems). RKFDA gives the smallest AUC in five problems, as also does the DFDA_5 model but, in general, all models except DFDA_0 appear to have a similar performance. We also observe that the best test AUC values in Table III are higher in almost all cases than the values computed for the same data sets using six state-of-the-art procedures in imbalanced classification and reported in [47]. In any case, note that the methods here are much stronger than the relatively weak decision tree classifiers used in [47].

The results in Table III are mostly descriptive. In order to achieve a more precise analysis, observe that considering all the 30 problems and their five train-test splits, we have a total of 150 AUC values for each one of the six models considered. This suggests that a more objective comparison can be achieved by applying a paired Wilcoxon signed-rank test for each model pair over these 150 AUC values, using Bonferroni corrections to compensate for multiple comparisons. The resulting p -values under the null hypothesis are shown in Table IV. As it could be expected from the previous discussion, the null hypothesis can only be rejected when the DFDA_0 model is compared against the others; this is not the case in all other comparisons and we can conclude that the performance of DFDA models with two or more hidden units is similar to that of RKFDA models.

We finally point out that in our experiments in this and Section VII we have used the MLPRegressor class in scikit-learn [48] for our implementation of DFDA networks, which makes its programming and execution very

TABLE III
TEST AUC FOR RKFDA AND THE 0- AND 2- TO 5-HIDDEN LAYER DFDA MODELS ON THE TWO-CLASS PROBLEMS

	RKFDA	DFDA_0	DFDA_2	DFDA_3	DFDA_4	DFDA_5
abalone9-18	88.67±7.13 (6)	95.44±2.13 (1)	91.68±6.21 (3)	90.67±4.34 (5)	90.95±4.77 (4)	91.72±5.46 (2)
abalone19	69.01±8.86 (6)	70.61±8.43 (5)	71.71±9.49 (4)	73.59±6.66 (2)	73.40±6.31 (3)	73.96±7.62 (1)
ec4	99.52±0.63 (4)	99.84±0.32 (1)	99.68±0.30 (2)	98.90±1.09 (5)	99.53±0.46 (3)	89.13±19.57 (6)
ec-0-1-3-7_vs_2-6	95.62±6.15 (1)	87.63±22.01 (3)	81.80±33.66 (5)	86.51±22.63 (4)	78.11±34.32 (6)	91.07±13.92 (2)
ec-0-1-4-6_vs_5	95.10±6.22 (5)	90.67±9.52 (6)	98.08±2.65 (2)	97.88±3.11 (3)	98.75±1.41 (1)	96.63±4.88 (4)
ec-0-1-4-7_vs_2-3-5-6	96.97±3.13 (1)	91.50±9.70 (6)	94.23±6.59 (3)	93.05±5.11 (5)	95.89±2.56 (2)	93.47±6.33 (4)
ec-0-1-4-7_vs_5-6	96.38±4.48 (3)	95.22±5.18 (4)	97.92±1.49 (2)	94.12±6.36 (6)	94.70±5.92 (5)	98.05±1.63 (1)
ec-0-1_vs_5	97.16±3.72 (3)	92.73±8.38 (6)	98.52±1.17 (1)	97.39±1.37 (2)	93.07±9.30 (5)	94.09±6.35 (4)
ec-0-3-4-6_vs_5	98.11±2.15 (4)	87.57±9.41 (6)	98.24±1.79 (3)	98.78±2.11 (2)	97.70±2.25 (5)	98.92±1.52 (1)
ec-0-3-4-7_vs_5-6	97.78±3.80 (1)	94.19±8.08 (6)	96.25±6.64 (3)	95.30±6.21 (5)	96.75±4.65 (2)	95.72±4.23 (4)
ec-0-6-7_vs_5	94.88±4.25 (1)	92.88±4.85 (5)	93±7.97 (4)	94.25±3.90 (2)	93.12±4.94 (3)	90.88±8.45 (6)
glass2	67.61±22.70 (6)	81.53±11.72 (4)	87±9 (3)	89.22±7.33 (1)	78.69±19.07 (5)	88.06±8.94 (2)
glass4	97.18±3.06 (4)	95.52±2.44 (6)	96.87±2.99 (5)	98.09±1.97 (1)	97.83±3.56 (2)	97.76±2.52 (3)
glass5	98.54±1.42 (3)	92.68±4.69 (6)	99.27±1.46 (1)	98.78±1.54 (2)	94.39±5.95 (5)	97.32±4.25 (4)
glass-0-1-4-6_vs_2	79.07±13.16 (5)	80.53±19.42 (3)	79.77±8.51 (4)	82.93±6.63 (1)	81.32±8.93 (2)	78.36±9.98 (6)
glass-0-1-6_vs_2	71.81±10.62 (5)	83.86±9.45 (1)	76.19±19.48 (3)	68.71±13.12 (6)	73.67±14.53 (4)	77.43±12.55 (2)
glass-0-1-6_vs_5	98.57±1.28 (4.5)	93.43±2.80 (6)	98.86±1.07 (2)	98.57±1.28 (4.5)	98.86±1.07 (2)	98.86±1.40 (2)
glass-0-4_vs_5	100±0 (3)	97.50±3.64 (6)	100±0 (3)	100±0 (3)	100±0 (3)	100±0 (3)
glass-0-6_vs_5	100±0 (3)	92.97±6.19 (6)	100±0 (3)	100±0 (3)	100±0 (3)	100±0 (3)
page-blocks-1-3_vs_4	100±0 (3)	96.38±0.95 (6)	100±0 (3)	100±0 (3)	100±0 (3)	100±0 (3)
shuttle-c0-vs-c4	100±0 (3)	100±0 (1.5)	89.98±19.99 (6)	100±0 (1.5)	99.98±0.03 (4)	99.92±0.15 (5)
shuttle-c2-vs-c4	100±0 (2)	100±0 (2)	99.20±1.60 (5)	99.20±1.60 (5)	100±0 (2)	99.20±1.60 (5)
vowel0	100±0 (3)	96.17±2.06 (6)	100±0 (3)	100±0 (3)	100±0 (3)	100±0 (3)
yeast4	85.46±3.24 (5)	86.55±2.21 (2)	85.68±10.21 (4)	86.69±7.26 (1)	86.49±5.44 (3)	84.96±12 (6)
yeast5	98.70±0.46 (4)	98.30±0.74 (6)	98.58±0.62 (5)	99.03±0.41 (3)	99.07±0.22 (2)	99.09±0.13 (1)
yeast6	91.88±8.17 (1)	91.28±8.29 (2)	89.85±8.56 (5)	89.65±8.99 (6)	91.03±6.12 (3)	90.61±8.92 (4)
yeast-0-5-6-7-9_vs_4	85.20±7.43 (2)	83.81±5.34 (6)	84.21±8.89 (4)	85.76±5.89 (1)	84.06±7.71 (5)	84.54±2.75 (3)
yeast-1-2-8-9_vs_7	73.68±5.15 (5)	73.93±7.32 (4)	77.17±12.61 (1)	72.10±13.30 (6)	76.57±11.92 (2)	76.48±13.03 (3)
yeast-1-4-5-8_vs_7	69.21±8.21 (3)	66.51±8.15 (6)	68.25±9.27 (5)	72.04±9.24 (2)	68.86±8.53 (4)	72.54±4.41 (1)
yeast-2_vs_8	83.72±4.47 (4)	76.80±5.13 (6)	80.11±11.32 (5)	84.21±12.10 (3)	87.59±8.27 (1)	86.92±13.25 (2)
rank mean	3.45	4.4833	3.4	3.2333	3.2333	3.2

TABLE IV
WILCOXON TESTS FOR TWO-CLASS PROBLEMS

	p-value	Hypothesis
RKFDA vs DFDA_0	0.00	Rejected
RKFDA vs DFDA_2	0.32	Not rejected
RKFDA vs DFDA_3	0.60	Not rejected
RKFDA vs DFDA_4	0.52	Not rejected
RKFDA vs DFDA_5	0.49	Not rejected
DFDA_0 vs DFDA_2	0.00	Rejected
DFDA_0 vs DFDA_3	0.00	Rejected
DFDA_0 vs DFDA_4	0.00	Rejected
DFDA_0 vs DFDA_5	0.00	Rejected
DFDA_2 vs DFDA_3	0.88	Not rejected
DFDA_2 vs DFDA_4	0.61	Not rejected
DFDA_2 vs DFDA_5	0.57	Not rejected
DFDA_3 vs DFDA_4	0.65	Not rejected
DFDA_3 vs DFDA_5	0.83	Not rejected
DFDA_4 vs DFDA_5	0.37	Not rejected

easy. We have also used `scikit-learn`'s routines and pipelines to implement the data scaling and cross-validation-based hyperparameterization. As for RKFDA, we have based our implementation on the `numpy` and `scipy` routines for eigenvalue computations, matrix inversion, and matrix multiplication. Thus, while the execution of general Python code may imply some computational overheads, the numerically heavier parts of our algorithms rely on a computationally efficient core. We have run our programs on a Fujitsu Primergy RX2540 server with 512 GB of RAM memory and Xeon E5-2640v4 processors at 2.4 GHz. Recall that the code used to implement RKFDA in our experiments is available at a <https://github.com/daviddiazvico?tab=repositoriesGitHub> repository.

VII. DFDA VERSUS RKFDA ON LARGE SCALE PROBLEMS

Large scale problems have an obvious importance in applications, particularly so when imbalanced and/or multiclass data sets are considered, but they seem not have been widely discussed in the literature [49]. We will compare here the performance of RKFDA and DFDA on two different data subsets with a number of moderate to large problems, some of them multiclass and/or imbalanced. The first set is taken from the <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/data> sets section of the LIBSVM web site. To determine the optimal values of the hyperparameters λ and γ , we will use the predefined train, validation, and test splits available for the `dna`, `ijcnn1`, `letter`, `satimage`, and `shuttle` data sets. The remaining data sets have only train and test splits and for them, we will hyperparameterize λ and γ by tenfold stratified cross validation on the train subset. The second comparison is made of the seven data sets with at least 1000 patterns among the 13 considered in the original RKFDA paper by Mika *et al.* [17], namely, `banana`, `German`, `image`, `ringnorm`, `splice`, `twonorm`, and `waveform`. These data sets are available in several web sites; we have used those at <http://theoval.cmp.uea.ac.uk/~cc/matlab/default.html#benchmarks>. In [17], RKFDA is compared against other machine learning algorithms using as the performance measure the accuracy of a threshold-based Fisher classifier (all problems have only two classes) which is chosen applying a 1-D linear support vector machine (SVM) classifier on

TABLE V
TEST AUC FOR RKFDA AND THE 0- AND 2- TO 5-HIDDEN LAYER MLPs ON THE LARGE SIZE PROBLEMS

	RKFDA	DFDA_0	DFDA_2	DFDA_3	DFDA_4	DFDA_5
a4a	84.33 (6)	84.66 (1)	84.56 (5)	84.63 (3)	84.63 (3)	84.63 (3)
a8a	85.16 (4)	84.80 (6)	85.08 (5)	85.48 (1)	85.43 (2)	85.41 (3)
dna	99.35 (1)	98.79 (6)	99.19 (3)	99.18 (4.5)	99.18 (4.5)	99.22 (2)
ijcnn1	99.07 (5)	87.74 (6)	99.42 (1)	99.37 (2)	99.20 (4)	99.36 (3)
letter	99.98 (1)	95.63 (6)	99.86 (5)	99.94 (4)	99.96 (2)	99.96 (3)
pendigits	99.97 (1)	97.54 (6)	99.97 (2)	99.95 (4)	99.94 (5)	99.97 (3)
satimage	78.28 (6)	93.26 (5)	97.66 (1)	94.43 (3)	94.38 (4)	97.64 (2)
shuttle	99.92 (4)	86.22 (6)	99.83 (5)	99.95 (3)	99.98 (1)	99.96 (2)
usps	99.77 (1)	98.30 (6)	99.74 (2)	99.66 (3.5)	99.66 (3.5)	99.66 (5)
w7a	92.76 (4)	90.23 (6)	92.55 (5)	93.73 (3)	95.22 (1.5)	95.22 (1.5)
w8a	94.68 (5)	89.79 (6)	96.30 (4)	99.07 (1)	98.82 (2)	98.73 (3)
banana	86.52±1.58 (5)	55.44±2.67 (6)	90.48±1.19 (3)	90.59±1.28 (1)	90.58±1.25 (2)	90.39±1.25 (4)
german	71.78±1.58 (1)	71.59±1.48 (3)	71.66±1.55 (2)	71.30±2.23 (4)	70.15±2.31 (6)	70.90±1.57 (5)
image	96.14±0.61 (2)	82.47±1.02 (6)	95.20±1.05 (5)	96.38±0.62 (1)	95.64±0.80 (4)	95.68±0.83 (3)
ringnorm	96.24±1.13 (5)	76.78±1.73 (6)	98.01±0.70 (3)	98.07±0.62 (1)	98.03±0.67 (2)	97.91±0.72 (4)
splice	90.69±1.25 (5)	84.43±1.09 (6)	93.65±0.96 (1)	93.53±0.92 (2)	93.29±0.80 (4)	93.33±0.99 (3)
twonorm	97.78±0.65 (6)	97.92±0.60 (2)	97.92±0.59 (3.5)	97.92±0.60 (5)	97.92±0.61 (3.5)	97.93±0.58 (1)
waveform	91.08±1.54 (4)	82.96±1.85 (6)	91.71±1.46 (1)	91.23±1.51 (2)	91.14±1.56 (3)	89.99±1.55 (5)
rank mean	3.6667	5.2778	3.1389	2.6667	3.1667	3.0833
combined	–	89.37 (5)	92.96 (1)	92.28 (3)	92.48 (2)	92.25 (4)
covtype.binary	–	55.74 (5)	59.32 (4)	61.12 (2)	61.91 (1)	59.83 (3)
skin_nonskin	–	97.80 (5)	99.26 (3)	99.87 (1)	99.81 (2)	99.16 (4)
rank mean	–	5	2.6667	2	1.6667	3.6667

the RKFDA outputs. In order to have a simpler comparison with our other results, we have compared the performance of RKFDA and DFDA using the test AUC values of our score-based classifiers.

Their dimensions, train sample sizes, number of classes, and imbalance ratios (defined now as the ratio between the maximum class size and the minimum one) are given in Table II. While some data sets are rather small (1000 patterns in *german*), others are quite large, particularly *covtype.binary*, with 581012 patterns. The maximum number of classes is 26 in *letter* and imbalance ratios go from near 1 in several data sets to the very large one for *shuttle*.

We will again compare here the performance of RKFDA models against DFDA models comprising 0–5 hidden layers of width 100, following the procedure of Section VI; recall that we will scale DFDA inputs to 0 mean and 1 standard deviation featurewise, while, for RKFDA models, the inputs will be scaled also featurewise to a $[0, 1]$ range.

Optimal λ and γ values for the LIBSVM data sets are obtained as those giving a higher geometric macro-AUC in the validation subsets and these values are used to build the final models over the train or train plus validation subsets when the latter exist, and to compute the multiclass geometric macro-AUC over the test subsets. The data sets used by Mika *et al.* have 100 predefined train-test splits; as in [17], we use the first five splits for hyperparameter tuning and then report the average and the standard deviation of the test AUCs over the 100 splits (recall that the LIBSVM data sets only have a single test set and no standard deviation can be computed).

These final AUC values are given in Table V; we also omit again the results for the DFDA_1 classifier. For convenience, it is divided into three parts for the medium size LIBSVM data sets, the Mika data sets and, finally, the large

LIBSVM data sets. While in several problems, DFDA and RKFDA models give similar geometric macro-AUC values, in *satimage*, *w7a*, *w8a*, *banana*, *ringnorm*, and *splice*, the AUC values of RKFDA are lower, while still in other problems, namely, *combined*, *covtype.binary*, and *skinnonskin*, we have not been able to properly hyperparameterize RKFDA on them. In fact, RKFDA and DFDA training required about the same times on the rather small two-class problems but RKFDA training took much longer times as the data set sizes increased. In particular, given that we were not able to obtain RKFDA hyperparameters for the three largest data sets, we do not give RKFDA’s AUC values for them at the table’s bottom. As before, each method’s ranking is shown for each problem in the table; note the rankings of the first two groups run from 1 to 6 but those at its bottom going from 1 to 5. Here, also, the DFDA_0 model seems to give worse results and the DFDA_2 also appears to have a worse performance than the models with three or more hidden layers; note also that the DFDA_3 seems to have a slight edge over the others.

As in the two-class case, the results in Table V and, particularly, its rankings, also have here a descriptive nature. The number of observations of the models for which proper hyperparameters are obtained for RKFDA is now 18, a relatively low number at the edge of what is usually taken as to justify a more precise Wilcoxon-based comparison between the DFDA and RKFDA models. Nevertheless, even with these caveats in mind, we have also applied here a paired Wilcoxon signed-rank tests for the first two problem sets in Table V. Its results appear in Table VI and, as it can be seen, if the Wilcoxon approximation hypotheses hold, the null hypothesis could be rejected when comparing the DFDA_0 model against the others. On the other hand, when RKFDA is compared with the DFDA_3 and DFDA_4, we can reject the null hypothesis

TABLE VI
WILCOXON TESTS FOR THE LARGE SIZE PROBLEMS

	p-value	Hypothesis
RKFDA vs DFDA_0	0.00	Rejected
RKFDA vs DFDA_2	0.13	Not rejected
RKFDA vs DFDA_3	0.01	Rejected
RKFDA vs DFDA_4	0.04	Rejected
RKFDA vs DFDA_5	0.08	Not rejected
DFDA_0 vs DFDA_2	0.00	Rejected
DFDA_0 vs DFDA_3	0.00	Rejected
DFDA_0 vs DFDA_4	0.00	Rejected
DFDA_0 vs DFDA_5	0.00	Rejected
DFDA_2 vs DFDA_3	0.59	Not rejected
DFDA_2 vs DFDA_4	0.87	Not rejected
DFDA_2 vs DFDA_5	0.74	Not rejected
DFDA_3 vs DFDA_4	0.05	Rejected
DFDA_3 vs DFDA_5	0.38	Not rejected
DFDA_4 vs DFDA_5	0.72	Not rejected

at the 0.05 level and could also do so at the 0.1 level when it is compared with DFDA_5 (we could also reject it when comparing the DFDA_3 and DFDA_4 models). Therefore, while the application of a Wilcoxon test may not be rigorously justified, the results in Tables V and VI point to a slightly better performance of some of the DFDA models over the RKFDA ones on larger sample size problems.

VIII. CONCLUSION

In this paper, we have reviewed the classical (FDA) and kernel (RKFDA) approaches to Fisher’s discriminant analysis following the analysis of Zhang *et al.* [18]. We have emphasized the computational complexity of RKFDA, which is cubic on the sample’s size N and, hence, could become prohibitive for large data sets. Aiming to overcome this, we have proposed DFDA networks, a simple yet very powerful DNN alternative to achieve a nonlinear form of FDA adapting the least squares formulation of FDA proposed in [34] and [18].

We have compared DFDA and RKFDA on a large number of highly imbalanced two-class problems of relatively small sizes as well as in a number of two-class and multiclass problems with substantially larger sizes and some of them with large class imbalances. Our experimental results show that while RKFDA and the deep DFDA models give similar results on the smaller data sets (excluding, of course, the linear model DFDA_0), the deeper DFDA models seem to improve the purely classification performance of RKFDA on the large data sets: although the application of a Wilcoxon test is not fully justified, if done, we could reject the null hypothesis at the 0.05 level in two cases and could do so at the 0.1 level in another. Moreover, when computational complexity considerations are taken into account, DFDA models seem to clearly beat the RKFDA ones, whose training times on the larger data sets are much higher (and even failed to finish in some cases).

We can thus conclude that the new deep DFDA networks we propose here are a simple yet powerful alternative to the more established regularized kernel-based RKFDA models in general classification problems and particularly so in imbalanced ones. We have seen both approaches to have a similar (and quite good) performance on highly imbalanced, relatively small two-class problems; on larger problems, DFDA networks appear to perform better than RKFDA models from a pure

classification point of view, while being clearly superior from a computational perspective. In fact, the need of handling kernel matrices makes the direct application of RKFDA models quite costly, putting large but clearly not big data problems out of their reach (a situation not dissimilar to what other kernel-based methods such as SVMs face over large data sets).

In any case, there are clear ways to improve the performance of DFDA models that seem precluded to RKFDA ones, of which we mention three. First, better results are to be expected if problem-tailored numbers of hidden layers and units are chosen instead of the fixed architectures used here. Second, the fully connected networks we have considered can be combined or substituted with any other of the many processing layer proposals that have been made for deep networks. A clear example is convolutional layers: they directly fit in the proposed deep Fisher approach and could certainly lead to an improved classification performance on problems such as image classification whose inputs have a spatial structure upon which convolutional filters act naturally.

Finally, we point out that, even after a nonlinear pre-processing, the Fisher criterion may not lead to the strongest classifiers. In fact, in the linear case, the cross-entropy loss used in logistic regression often produces better models. On the other hand, the structure induced by the Fisher criterion on the last hidden layer of a DFDA network yields new features that a more powerful classifier could take advantage of. This naturally suggests that one could add the least squares DFDA loss as a companion to another, problem-specific loss that then takes advantage of the within-class concentration and between class separation structure of the Fisher-like representation on the last hidden layer.

While that might have been rather difficult a few years ago, the most widely used deep net frameworks (Torch, TensorFlow, Theano, MXNET, or CNTK) provide backpropagation gradients automatically through their network “compilation” procedures. This means that more general loss functions than square error or cross entropy can be considered without having to program their gradients “by hand,” as it was needed in the early 2000s. We can thus add the DFDA loss function into any strong deep model (such as AlexNet or VGG-16 for image processing) in a way that can enhance its performance. We are currently pursuing these and other related research goals.

ACKNOWLEDGMENT

The authors would like to thank the Centro de Computación Científica (CCC) at Universidad Autónoma de Madrid (UAM) for the use of their facilities.

REFERENCES

- [1] B. Krawczyk, “Learning from imbalanced data: Open challenges and future directions,” *Prog. Artif. Intell.*, vol. 5, no. 4, pp. 221–232, 2016.
- [2] N. Japkowicz and S. Stephen, “The class imbalance problem: A systematic study,” *Intell. Data Anal.*, vol. 6, no. 5, pp. 429–449, 2002.
- [3] H. He and E. A. Garcia, “Learning from imbalanced data,” *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.
- [4] N. Japkowicz, C. Myers, and M. A. Gluck, “A novelty detection approach to classification,” in *Proc. 14th Int. Joint Conf. Artif. Intell.*, vol. 1, Aug. 1995, pp. 518–523.
- [5] M. Kubat and S. Matwin, “Addressing the curse of imbalanced training sets: One-sided selection,” in *Proc. 14th Int. Conf. Mach. Learn.*, vol. 97, 1997, pp. 179–186.

- [6] T. Fawcett and F. J. Provost, "Adaptive fraud detection," *Data Mining Knowl. Discovery*, vol. 1, no. 3, pp. 291–316, 1997.
- [7] P. M. Domingos, "MetaCost: A general method for making classifiers cost-sensitive," in *Proc. 5th ACM SIGKDD int. conf. Knowl. Discovery Data Mining*, vol. 99, Aug. 1999, pp. 155–164.
- [8] B. Tang and H. He, "GIR-based ensemble sampling approaches for imbalanced learning," *Pattern Recognit.*, vol. 71, pp. 306–319, Nov. 2017.
- [9] X. Liu, J. Wu, and Z. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Trans. Syst., Man, Cybern. B*, vol. 39, no. 2, pp. 539–550, Apr. 2009.
- [10] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002.
- [11] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Trans. Sys. Man, Cybern. C, Appl. Reviews*, vol. 42, no. 4, pp. 463–484, Jul. 2012.
- [12] Z. Zhou and X. Liu, "Training Cost-Sensitive neural networks with methods addressing the class imbalance problem," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 1, pp. 63–77, Jan. 2006.
- [13] A. R. Webb and D. Lowe, "The optimised internal representation of multilayer classifier networks performs nonlinear discriminant analysis," *Neural Netw.*, vol. 3, no. 4, pp. 367–375, 1990.
- [14] K. Fukunaga, *Introduction to statistical pattern recognition*, Boston, MA, USA: Academic, 1990.
- [15] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. New York, NY, USA: Wiley, 2000.
- [16] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 7, pp. 711–720, Jul. 1997.
- [17] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, A. J. Smola, and K. R. Müller, "Invariant feature extraction and classification in kernel spaces," in *Proc. Adv. Neural Inf. Process. Syst.*, 1999, pp. 526–532.
- [18] Z. Zhang, G. Dai, C. Xu, and M. I. Jordan, "Regularized discriminant analysis, ridge regression and beyond," *J. Mach. Learning Res.*, vol. 11, pp. 2199–2228, Aug. 2010.
- [19] J. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "Face recognition using kernel direct discriminant analysis algorithms," *Trans. Neural Netw.*, vol. 14, no. 1, pp. 117–126, Jan. 2003.
- [20] V. Sydorov, M. Sakurada, and C. H. Lampert, "Deep fisher kernels—end to end learning of the fisher kernel GMM parameters," in *Proc. Conf. Comput. Vision Pattern Recognit.*, Washington, DC, USA, Jun. 2014, pp. 1402–1409.
- [21] J. Yang, A. F. Frangi, J.-Y. Yang, D. Zhang, and Z. Jin, "KPCA plus LDA: A complete kernel fisher discriminant framework for feature extraction and recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 2, pp. 230–244, Feb. 2005.
- [22] J. Bergstra *et al.*, "Theano: A CPU and GPU math expression compiler," in *Proc. Python sci. Comput. conf. (SciPy)*, Jun. 2010, vol. 4, no. 3, pp. 1–25.
- [23] F. Seide and A. Agarwal, "CNTK: Microsoft's open-source deep-learning toolkit," in *Proc. 22nd ACM SIGKDD Inter. Conf. Knowledge Discovery Data Mining*, 2016, p. 2135.
- [24] R. Collobert and K. Kavukcuoglu, "Torch7: A matlab-like environment for machine learning," in *Proc. BigLearn, NIPS Workshop*, vol. 5, no. 10, Dec. 2011, pp. 1–6.
- [25] T. Chen *et al.* (2015). "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems." [Online]. Available: <https://arxiv.org/abs/1512.01274>
- [26] Google. *Tensorflow, An Open Source Software Library for Machine Intelligence*. Accessed: Mar. 2019. [Online]. Available: <https://www.tensorflow.org/>
- [27] C. S. Cruz and J. R. Dorronsoro, "A nonlinear discriminant algorithm for feature extraction and data classification," *IEEE Trans. Neural Netw.*, vol. 9, no. 6, pp. 1370–1376, Nov. 1998.
- [28] A. Stuhlsatz, J. Lippel, and T. Zielke, "Feature extraction with deep neural networks by a generalized discriminant analysis," *IEEE Trans. Neural Netw. Learn. Sys.*, vol. 23, no. 4, pp. 596–608, Apr. 2012.
- [29] M. Dorfer, R. Kelz, and G. Widmer. (Nov. 2015). "Deep linear discriminant analysis." [Online]. Available: <https://arxiv.org/abs/1511.04707>
- [30] L. Wu, C. Shen, and A. van den Hengel, "Deep linear discriminant analysis on fisher networks: A hybrid architecture for person re-identification," *Pattern Recognit.*, vol. 65, pp. 238–250, May 2017.
- [31] Q. Tian, T. Arbel, and J. J. Clark, "Deep LDA-pruned nets for efficient facial gender classification," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit. (CVPR) Workshops*, Honolulu, HI, USA, Jul. 2017, pp. 512–521.
- [32] K. Lee and J. Kim, "Font Size: On the equivalence of linear discriminant analysis and least squares," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 2736–2742.
- [33] C. H. Park and H. Park, "A relationship between linear discriminant analysis and the generalized minimum squared error solution," *SIAM J. Matrix Anal. Applications*, vol. 27, no. 2, pp. 474–492, Jun. 2005.
- [34] J. Ye, "Least squares linear discriminant analysis," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 1087–1093.
- [35] D. Cai, X. He, and J. Han, "SRDA: An efficient algorithm for large-scale discriminant analysis," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 1, pp. 1–12, Jan. 2008.
- [36] D. Díaz-Vico A. Omari, A. Torres-Barrán, and J. R. Dorronsoro, "Deep fisher discriminant analysis," in *Proce. 14th Int. Work-conf. Artif. Neural Netw.*, 2017, pp. 501–512.
- [37] J. H. Friedman, "Regularized discriminant analysis," *J. Amer. Statist. Assoc.*, vol. 84, no. 405, pp. 165–175, 1989.
- [38] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.
- [39] C. Ferri, J. Hernandez-Orallo, and M. Salido, "Volume under the ROC surface for multi-class problems," in *Proc. 14th Eur. Conf. Machine Learn.*, 2003, pp. 108–120.
- [40] F. Provost and P. Domingos, "Tree induction for probability-based ranking," *Machine Learn.*, vol. 52, no. 3, pp. 199–215, Sep. 2003.
- [41] G. Tsoumakas, I. Katakis, and I. P. Vlahavas, "Mining multi-label data," in *Data Mining Knowl. Discovery Handbook*, 2nd ed. New York, NY, USA: Springer, 2010, pp. 667–685.
- [42] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
- [43] D. J. Hand and R. J. Till, "A simple generalisation of the area under the ROC curve for multiple class classification problems," *Machine Learn.*, vol. 45, no. 2, pp. 171–186, Nov. 2001.
- [44] D. Díaz-Vico, A. R. Figueiras-Vidal, and J. R. Dorronsoro, "Deep MLPs for imbalanced classification," in *Proc. Int. Joint Conf. Neural Netw.*, 2018, pp. 1–7.
- [45] Y. Yang, "An evaluation of statistical approaches to text categorization," *Inf. Retrieval*, vol. 1, nos. 1–2, pp. 69–90, 1999.
- [46] J. Alcalá-Fdez *et al.* "Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework," *J. Multiple-Valued Logic Soft Comput.*, vol. 17, nos. 2–3, pp. 255–287, 2011.
- [47] S. González S. García, M. Lázaro A. R. Figueiras-Vidal, and F. Herrera, "Class switching according to nearest enemy distance for learning from highly imbalanced data-sets," *Pattern Recognit.*, vol. 70 pp. 12–24, 2017.
- [48] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *J. achine Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
- [49] S. Wang and X. Yao, "Multiclass imbalance problems: Analysis and potential solutions," *IEEE Trans. Sys. Man Cybern., B (Cybern.)*, vol. 42, no. 4, pp. 1119–1130, 2012.



David Díaz-Vico received the M.Sc. degree in mathematics and the M.S.E. degree in computer science from the Universidad Autónoma de Madrid, Madrid, Spain, in 2012, where he is currently pursuing the Ph.D. degree in computer science.

He was a Data Scientist with Accenture Analytics, Madrid, Telefónica Research and Development, Madrid, and Instituto de Ingeniería del Conocimiento (IIC), Madrid, for more than 8 years. He has authored several papers. He holds patents in machine learning and applications.



José R. Dorronsoro received the Ph.D. degree with Washington University in St. Louis, St. Louis, MO, USA, in 1981.

He is currently a Professor of computer science with the Universidad Autónoma de Madrid, Madrid, Spain. He has directed eight Ph.D. theses and has been a Leader of a large number of research and innovation projects. He is also a Senior Scientist with the Instituto de Ingeniería del Conocimiento (IIC), Madrid, where he leads IIC's research and innovation on the application of machine learning to areas such as renewable energy. He has authored more than 100 scientific papers in mathematical analysis, machine learning, and applications.

Deep Fisher Discriminant Analysis

David Díaz-Vico^(*), Adil Omari, Alberto Torres-Barrán,
and José Ramón Dorronsoro

Dpto. Ing. Informática and Instituto de Ingeniería del Conocimiento,
Universidad Autónoma de Madrid, Madrid, Spain
`david.diazv@estudiante.uam.es`

Abstract. Fisher Discriminant Analysis' linear nature and the usual eigen-analysis approach to its solution have limited the application of its underlying elegant idea. In this work we will take advantage of some recent partially equivalent formulations based on standard least squares regression to develop a simple Deep Neural Network (DNN) extension of Fisher's analysis that greatly improves on its ability to cluster sample projections around their class means while keeping these apart. This is shown by the much better accuracies and g scores of class mean classifiers when applied to the features provided by simple DNN architectures than what can be achieved using Fisher's linear ones.

Keywords: Linear Discriminant Analysis · Deep Neural Networks · Non-linear classifiers

1 Introduction

Fisher' Linear Discriminant Analysis (FLDA from now on) is a very well known linear dimensionality reduction/feature extraction technique that, while able to provide useful data representations, does not intend, in principle, to solve a given classification problem and, thus, it has known only a limited use as a tool to build classifiers. There may be two main reasons for this. The first one is its linear nature. In fact, while quite attractive, its main goal of concentrating the projected features around their class means while keeping those means apart can, for most problems, only be partially achieved by FLDA's linear projections. Moreover, in order to build a powerful classifier, we would most likely need to apply a non-linear classifier to the FLDA features, but this combination of a linear projection followed by a non-linear classifier may at best be only competitive with the direct application of the non-linear classifier over the initial features.

In any case, FLDA has been successfully applied in a number of problems, most notably on face recognition where the original Fisher Face method [1] has been progressively improved to become the state of the art in this area. A natural idea is thus to somehow extend FLDA to a nonlinear procedure by applying it after some non-linear pre-processing of the original features. This is the goal of Kernel Discriminant Analysis (KDA, [12]), which addresses binary problems and

where the well known reduction for such problems of FLDA to a linear regression problem [4, Chap. 5], is extended into a kernel setting.

As just said, KDA only is available in principle for binary problems. A different non-linear extension that works for multiclass problems was proposed in [8, 14], where FLDA is applied on the nonlinear features obtained after processing the original features by the hidden layers of a standard feed-forward multilayer perceptron (MLP). For this, the non-linear z features are first obtained on the MLP's last hidden layer as $z = f(x, W)$, where W denotes the MLP weights and biases up to the last hidden layer and $f(x, W)$ the effect of the MLP forward pass on the original features x . Then, FLDA's standard criterion function is used on these z to get FLDA's projecting matrix A by minimizing one of the several criterion functions J proposed in FLDA. Thus, we can view the overall cost function $J(A, W)$ as depending separately on the FLDA's projection matrix A and on the MLP's weight and bias set W , which suggests to optimize $J(A, W)$ alternating the minimization on W and A . More precisely, for a given W_k and $z^k = f(x, W_k)$, we first derive the A_k matrix by minimizing $J(A, W_k)$ by FLDA's standard eigen-procedure. Then, the new W_{k+1} are derived minimizing $\mathcal{J}(W) = J(A_k, W)$; as shown in [8], the gradient $\nabla_W \mathcal{J}$ can be explicitly computed by backpropagation. Notice that this ensures $J(A_k, W_{k+1}) < J(A_k, W_k) < J(A_{k-1}, W_k)$ and this alternating two-step process can be iterated towards a minimum (A^*, W^*) of J . As in [8, 14], we shall refer to this procedure as Non-linear Discriminant Analysis (NLDA).

While in principle any number of hidden layers could be considered, only a single hidden layer was used in [8, 14], as was customarily done before the advent of deep neural networks (DNN). These have provided two main insights. The first one is a better understanding of network initialization plus efficient minimization and regularization procedures, which have made largely routine the previously near impossible training of many layered networks. The second one is the availability of symbolic gradient computation in platforms such as Theano [2] or TensorFlow [9] that make it possible the consideration of cost functions much more general than the square error or cross entropy usually applied in neural network-based regression or classification.

Both could be applied to improve on NLDA, either by keeping the alternating minimization of $J(A, W)$ but working with deeper networks or, simply by applying symbolic differentiation on A, W directly to the joint $J(A, W)$ cost function. Here, however, we will follow a much more direct approach by taking advantage of the results in [13, 15, 16], where a link is established between a concrete formulation of FLDA and a related Least Squares Regression (LSR) problem with a particular, class-based target choice. We shall make extensive use of this approach which we will call Least Squares Discriminant Analysis (LSDA). More precisely, it is shown in [15, 16] that there is an isometry between projections derived from a specific FLDA formulation and those derived from the solution of the LSR problem. In turn, this implies that if distance based classifiers such as k -Nearest Neighbors or (as done here) minimum class-mean distances are used, either a renormalized FLDA or LSR approaches result in equivalent classifiers.

Once the previous set up is available, it is straightforward to carry the LSR problem into a DNN setting, working with the same targets as in the linear case but which now are to be approximated by the outputs of a suitable DNN. This is the approach we shall follow here and, besides a short, self-contained presentation of the LSR and FLDA equivalence in [15, 16] our contributions are the following:

- The proposal of Deep Fisher Discriminant Analysis, DFDA, along the lines just summarized.
- A comparison of DFDA with classical FLDA over several, large size, binary and multiclass datasets, that shows a much better performance of DFDA.

The paper is organized as follows. In Sect. 2 we shall review classical FLDA as well as the distance-based classifier equivalence established in [15, 16] between classical FLDA and a concrete LSR problem. Deep Fisher Discriminant Analysis is introduced in Sect. 3 and in Sect. 4 we will compare its performance with that of classical LFDA over several relatively large multiclass and, in some cases, imbalanced problems. As we shall see, the accuracies and g scores of the DFDA classifiers are substantially better. Finally the paper closes with a brief discussion and pointers to further work.

2 Fisher’s Linear Discriminant Analysis and Least Squares Counterparts

2.1 Fisher’s Linear Discriminant Analysis

We first briefly review classical FLDA. As mentioned, its goal is to linearly project the original patterns in such a way that these projections are close to their class means while these class means are kept apart. Several criterion functions can be used for this goal and many of them are in fact equivalent; see [6, Sect. 10.2]. Here we will seek to maximize the trace criterion

$$g(W) = \text{trace}(s_T^{-1} s_B) = \text{trace} \left((A^t S_T A)^{-1} (A^t S_B A) \right), \quad (1)$$

where A is the projection matrix, S_B and S_T denote the between-class and total covariance matrices respectively of the sample patterns and s_B and s_T are their counterparts for the projections $z = Ax$; see [16], Subsect. 2.2 for more details. Solving $\nabla_A g = 0$ leads to

$$0 = -2S_T A s_T^{-1} s_B s_T^{-1} + 2S_B A s_T^{-1}$$

i.e., $S_T^{-1} S_B A = A s_T^{-1} s_B$ or, up to an invertible transformation of A (which won’t change the cost function $g(W)$), to

$$S_T^{-1} S_B A = A \Lambda, \quad (2)$$

with Λ the non-zero eigenvalues of $S_T^{-1} S_B$ (and of $s_T^{-1} s_B$). Thus, for such an A we have

$$g(A) = \text{trace}(s_T^{-1} s_B) = \text{trace } \Lambda = \lambda_1 + \dots + \lambda_q,$$

which we maximize by sorting the eigenvalues in Λ in descending order and selecting the $q = \min\{d, C - 1\}$ largest ones and some conveniently normalized associated eigenvectors; here d is pattern dimension and C the number of classes; q is then the rank of S_B . Notice that the minimizer of (1) is not uniquely defined, and some normalization has to be introduced; an usual choice is to impose $A^t S_T A = I_q$.

In some problems S_T may be ill conditioned and not have full rank. One possibility in this case is to use the Moore-Penrose inverse of S_T ; another, and the one we follow here, is Regularized Discriminant Analysis [5], where we work with $S_t + \lambda I$ for an appropriate $\lambda > 0$.

2.2 Least Squares Regression and Fisher's Linear Discriminant Analysis

It is a very well known result [4] that for 2-class problems, a solution to FLDA can be obtained solving a Least Squares Regression (LSR) problem

$$\min \frac{1}{2} \|Y - \mathbf{1}_n w_0 - Xw\|^2 \quad (3)$$

where $\mathbf{1}_n$ is the all ones vector, X is the $n \times d$ data matrix and Y is an appropriate target matrix defined by setting $Y_p = n/n_1$ if x_p belongs to class 1 and $Y_p = -n/n_2$ if x_p belongs to class 2. If m_1 and m_2 denote the class means on the original features, it can be then checked that $w = S_T^{-1}(m_1 - m_2)$. Since now $S_B = (m_1 - m_2)(m_1 - m_2)^t$, it follows that $S_T^{-1} S_B w = w\gamma$, with $\gamma = (m_1 - m_2)^t S_T^{-1}(m_1 - m_2)$, i.e., w is an eigen-solution of (2) and, hence, a dilation of a FLDA's projection vector computed as in the previous subsection.

There have been several attempts to carry this result to a multiclass setting. Among the most successful ones are the proposals by Park and Park [13] and the somewhat simpler one in Zhang *et al.* [16], which we follow here and briefly explain next in a much more concise way.

Consider again the LSR problem (3), where the target matrix Y to be adequately chosen. For simplicity we assume that S_T is regular; if not, we can simply replace it with $S_T + \lambda I$ for some $\lambda > 0$. The optimal LSR solution is then

$$w = S_T^{-1} X^t H Y,$$

where H is the **centering matrix**

$$H = I_n - \frac{1}{n} \mathbf{1}_n \mathbf{1}_n^t;$$

in particular, $X^t H$ is the $n \times d$ matrix whose p -th row equals $x_p^t - m^t$. The key assumption is now that we can choose a target matrix Y such that we can write S_B as

$$S_B = X^t H Y Y^t H X.$$

Assuming this, let us write $Q = X^t H Y$ (and, thus, $S_B = Q Q^t$); we show next how can we transform the solution $w = S_T^{-1} X^t H Y = S_T^{-1} Q$ of (3) into an eigen-solution A of (2). Consider the semidefinite positive matrix $R = Q^t S_T^{-1} Q$ whose

SVD decomposition is $R = \tilde{V}\tilde{\Gamma}\tilde{V}^t$. Assuming for simplicity that $\text{rank}(Q) = \text{rank}(S_B) = C - 1$, at least one of the diagonal elements in $\tilde{\Gamma}$ will be zero; reordering $\tilde{\Gamma}$ if necessary, we assume it to be the element in the (C, C) matrix entry. We can thus drop the last row and column of $\tilde{\Gamma}$ (that are 0) to get a $(C - 1) \times (C - 1)$ diagonal matrix Γ , and the last column of \tilde{V} to get a $C \times (C - 1)$ matrix V that verifies $V^tV = I_{C-1}$ and for which we can also write R as $R = V\Gamma V^t$.

Now it is easy to check that the pair $(A = wV\Gamma^{-1/2}, \Gamma)$ is an eigensolution of (2) with normalization $A^tS_TA = I_q$, for we have

$$\begin{aligned} S_T^{-1}S_BA &= S_T^{-1}QQ^twV\Gamma^{-1/2} = wQ^tS_T^{-1}QV\Gamma^{-1/2} = wRV\Gamma^{-1/2} \\ &= wV\Gamma V^tV\Gamma^{-1/2} = A\Gamma. \end{aligned}$$

In other words, if we choose Y adequately, from $Q = X^tHY$ we can derive the LSR solution w , the SVD decomposition (V, Γ) of $Q^tS_T^{-1}Q$ and the FLDA eigensolution $A = wV\Gamma^{-1/2}$. This combines Algorithm 4 in [16] and the discussion in its Sect. 6 to derive a FLDA solution A from the LSR solution w .

As mentioned, one thing to consider is the possibility of S_T being singular. This can be easily handled now by working with a Ridge Regression problem, i.e., solving for an appropriate $\lambda > 0$

$$\min \frac{1}{2} \|Y - \mathbf{1}_{nC} w_0^t - Xw\|^2 + \frac{\lambda}{2} \text{trace}(w^tw),$$

where $\mathbf{1}_{nC}$ denotes the $n \times C$ all ones matrix and w_0 is a C -dimensional vector.

2.3 Equivalence of Distance Classifiers

As in [16], let's consider the projection matrix $B = A\Gamma^{1/2} = wV$ instead of FLDA's standard A . It is easy to see that B is also an eigen-solution of (2) associated to the normalization $B^tS_TB = \Gamma$. Let $y = w_0 + w^tx$ and $\omega = B^tx$ be the LSR and B -eigen-projections of a pattern x , respectively. We then have

$$\|\omega - \omega'\|^2 = (x - x')^tBB^t(x - x') = (x - x')^twVV^tw^t(x - x') = \|y - y'\|^2.$$

Thus, any Euclidean distance-based classifier will give the same results when applied to the B -eigen-projections ω than when applied to the LSR ones y . This will be the case for a k -Nearest Neighbor classifier and also for the nearest class mean classifier

$$\delta_{NCM}(x) = \arg \min_c \|w^tx - w^t\bar{x}_c\| = \arg \min_c \|y - \bar{y}_c\|,$$

which we will use here. In other words, we can obtain a FLDA-like nearest class mean classifier directly from the LSR solution, without having to perform the eigen-analysis that FLDA requires. We will call this procedure, proposed in [15, 16], Least Squares Discriminant Analysis, LSDA, and take advantage of this in Sect. 3 to define our deep Fisher classifiers but, before that, we close this section with two examples of suitable target matrices Y .

2.4 Two Examples

The well known relationship between the LSR and FLDA solutions for 2 class problems mentioned above also follows easily from the previous discussion. In fact, using the target vector $Y_p = n/n_1$ if x_p is in class 1 and $Y_p = -n/n_2$ if x_p is in class 2, then we have $Q = X^tHY = m_1 - m_2$ and, hence, $S_B = (m_1 - m_2)(m_1 - m_2)^t = QQ^t$. Besides, $R = QS_T^{-1}Q = (m_1 - m_2)^t S_T^{-1}(m_1 - m_2) = \gamma$, with a trivial SVD decomposition $R = 1\gamma 1$ and, thus, we have here $A = w \cdot 1 \cdot \gamma^{-1/2} = S_T^{-1}(m_1 - m_2)\gamma^{-1/2}$ and $B = A\gamma^{1/2} = S_T^{-1}(m_1 - m_2) = w$, i.e., the w and y projections now coincide.

For the general multi-class case, it is shown in [16, Eq. (4)], that we can write

$$S_B = X^tHE\Pi^{-1}E^tHX = X^tHE\Pi^{-1/2}\Pi^{-1/2}E^tHX,$$

where Π is the diagonal matrix with $\Pi_{cc} = n_c$ and E is the $n \times C$ indicator matrix with rows e_p such that if x_p is in class c , $e_{pc} = 1$ and $e_{pc'} = 0$ for $c' \neq c$. Thus here we can take $Q = X^tHE\Pi^{-1/2}$ and it is also shown in [16, Eq. (25)], that we can write the LSR solution $w = S_T^{-1}X^tHY$ as

$$w = S_T^{-1}X^tHE\Pi^{-1/2} = S_T^{-1}Q$$

if we use $Y = HE\Pi^{-1/2}$ as the target matrix. It is now easy to see that for such Y and x_p in class c , we have $Y_{pc} = \frac{n-n_c}{n\sqrt{n_c}}$, and $Y_{pc'} = -\frac{\sqrt{n_c}}{n}$ otherwise. These are the targets we shall use in the next section.

3 Deep Fisher Discriminant Analysis

We have just argued how we can obtain for a general C class problem a nearest class-mean classifier equivalent to one acting on the B -based Fisher projections by performing the following steps:

1. For a given training set D_{tr} , solve for a data matrix X_{tr} , class indicator matrix E_{tr} and targets $Y_{tr} = HE_{tr}\Pi^{-1/2}$ the LSR problem

$$\min \frac{1}{2} \|Y_{tr} - \mathbf{1}_{nC} w_0^t - X_{tr}w\|^2,$$

obtaining the optimal $d \times C$ matrix w^* and C -dimensional vector w_0^* .

2. Compute the projections $y = w_0^* + (w^*)^t x$ for $x \in D_{tr}$ and their class means $\bar{y}_c = w_0^* + (w^*)^t \bar{x}_c$.
3. Assign an x in a test sample D_{ts} to the class whose mean the projection $y = w_0^* + (w^*)^t x$ it is closest to; that is, to the class c^* for which

$$c^* = \arg \min_c \{\|y - \bar{y}_c\|\}.$$

Now, a natural idea to extend this to a non linear setting is to perform the LSR computations on features z obtained by a non-linear processing $z = \Phi(x)$ of the original features x . An example of this is Kernel Discriminant Analysis,

KDA [12, 16], where a certain generalized eigenvalue problem involving the kernel matrix $K = ZZ^t$ is solved, with the matrix Z being $Z = \Phi(X)$; in particular the projections z are not needed explicitly as they enter the computations through a kernel k such that $z \cdot z' = \Phi(x) \cdot \Phi(x') = k(x, x')$. As it is often the case in kernel methods, handling the $n \times n$ matrix K can be too costly in large sample problems and some suitable low rank approximation would have to be used.

A simpler alternative, better suited in principle for large sample problems, is to derive the z features using a straightforward DNN extension of the previous linear setup; more precisely, in a DNN setting we would

1. Solve over a training set D_{tr} the LSR problem

$$\min \frac{1}{2} \|Y_{tr} - f(X_{tr}, \mathcal{W})\|^2$$

with Y_{tr} the previous training target matrix and the p -th row of the matrix $f(X_{tr}, \mathcal{W})_p$ is given by $f(X_{tr}, \mathcal{W})_p = f(x_p, \mathcal{W})$, where $f(x, \mathcal{W})$ is the transfer function of a deep network with linear outputs and overall weight set \mathcal{W} ; we thus obtain an optimal DNN weight set \mathcal{W}^* .

2. Compute the projections $y_p = f(x_p, \mathcal{W}^*)$ over D_{tr} .
3. Assign a new $x \in D_{ts}$ to the class whose mean the projection $y = f(x, \mathcal{W}^*)$ is closest to.

Writing $\mathcal{W}^* = (w_0^*, w^*, W^*)$ with w_0^*, w^* the linear output weights, these optimal w_0^*, w^* solve the LSR problem (3) over the last hidden layer features $z = \Phi(x, W^*)$, with Φ the DNN transfer function up to the last hidden layer. Thus, the class mean classifier of the full DNN is equivalent to a class mean classifier over some FLDA projections of the z patterns in the last hidden layer which, in turn, are also learned by tuning the W component of the overall weight \mathcal{W} . In other words, the DNN also performs a particular kind of representation learning, as in this case it learns in its last hidden layer new features that have been optimized to perform FLDA on them. We will call this Deep Fisher Discriminant Analysis, or DFDA.

As in the linear case, we may avoid singularity issues here by adding a regularization term, i.e., solving for instance

$$\min_{w, W} \frac{1}{2} \|Y - f(X, w_0, w, W)\|^2 + \frac{\lambda}{2} \text{trace}(w^t w + \widetilde{W}^t \widetilde{W}),$$

with \widetilde{W} the components of W excluding the biases at each hidden layer. This is the cost function we will use. Of course, in the deep case one may use other regularization terms for the W components of the overall weight structure \mathcal{W} (such as, for instance, dropout), but the term $\frac{\lambda}{2} \text{trace}(w^t w)$ should be kept in any case for the linear weights w .

4 Numerical Experiments

4.1 Datasets and Quality Measures

We will consider the datasets SensIT Vehicle (combined), dna, ijcnn1, letter, mnist, pendigits, satimage, shuttle, w7a, w8a and usps. All

Table 1. Train sample size, dimension, number of classes and ratio between the maximum and minimum class sizes for the considered datasets.

Problem	N. patterns	Dimension	N. classes	Class ratios
combined	78,823	100	3	2.156
dna	2,000	180	3	2.265
ijcnn1	49,990	22	2	9.301
letter	15,000	16	26	1.128
mnist	60,000	784	10	1.244
pendigits	7,494	16	10	1.085
satimage	4,435	36	6	2.583
shuttle	43,500	9	7	5,684.667
w7a	24,692	300	2	32.368
w8a	49,749	300	2	32.637
usps	7,291	256	10	2.203

of them also have separate, well defined train-test splits and, except `mnist`, are available on the [Datasets section](#) of the LIBSVM web site; for `mnist` we have used Scikit-Learn to fetch it from [mldata](#). We have put an emphasis in relatively large, multiclass datasets; in Table 1 we give their dimension, total number of train patterns, number of classes and their maximum class size ratios, i.e., the ratio of the maximum class size to the minimum one. Data sizes go from 2,000 (`dna`) to 78,823 (`combined`) and the number of classes ranges from 2 to 26 (`letter`); while some of them are quite balanced, others (`w7a`, `w8a` and particularly `shuttle`) present large class imbalances, having class size ratios $\gg 1$. Because of this, the main quality measure we will use for model evaluation will be the g -score, i.e., the geometric mean of the different class sensitivities:

$$g = \left(\prod_{c=1}^C S_c \right)^{1/C} = \left(\prod_1^C \frac{m_{cc}}{\sum_j m_{cj}} \right)^{1/C},$$

where m_{cj} is the (c, j) entry of the confusion matrix, that is, the number of class c patterns that are assigned to class j . The g -score measure is often used in imbalanced classification as it is more robust to markedly different class sizes than accuracy, easily achieved by assigning small class patterns to the largest class. Because of this we will also use g as the merit function for hyper-parameter selection of both linear and deep models. Nevertheless, we shall also report the accuracies $a = \frac{\sum_c m_{cc}}{\sum_{c,j} m_{cj}}$.

4.2 Deep Model Universal Approximation Capabilities

The goal when using deep versions of FLDA is to obtain better representations in the last hidden layer of which the final, Fisher-like, linear transform can

take advantage. In an extreme perspective, overfitting should concentrate each class around its mean while keeping these classes far apart. Once vanishing gradients are avoided and proper training is possible, the simplest way to overfit a dataset is to work with a deep enough network with rather large hidden layers, which ensures a large number of weights. We will do so here, building for each dataset four DFDA models having between 2 and 5 fully connected hidden layers with 100 units each but controlling overfit with a proper penalty. For most problems the number of weights approximately varies thus between 20,000 and 50,000; on the other hand, given its pattern dimension, for `mnist` the first hidden layer already has 78,400 weights. We stress that our main goal here is not to obtain top quality models; for instance, for `mnist`, convolutional networks would be needed for this and the fully connected layers be much larger than the ones considered here. Instead, our main goal here is to measure DFDA's performance and compare it against that of FLDA.

As a benchmark reference we will build an LSDA model for each dataset using the class `Ridge` in `scikit-learn`; for DFDA models we will use the `MLPRegressor` class also in `scikit-learn`. `MLPRegressor` only allows for deep MLPs with fully connected layers and L_2 penalization; we will use `relu` activations and the `adam` solver. This solver is a faster, more stable version of gradient descent but convergence may still be slow. Moreover, the targets Y_{tr} are rather small to begin with, so the convergence tolerance should also be small. Because of this we will work with a maximum number of 20,000 iterations and use a tolerance of 10^{-9} ; other solver parameters are left at their default values. A more powerful alternative could have been to work with a general DNN framework such as Keras [3] that has Theano or TensorFlow as backends and offers a much wider range of network architectures (including for instance convolutional layers) or penalties (L_1 or dropout). However, most of the problems considered do not lend themselves to, say, using convolutional layers and, on the other hand, the structural simplicity of `MLPRegressor` models results in a much faster training.

We will work with mini-batch sizes `min(200, num_patterns)`, i.e., the default for the `MLPRegressor` class. Therefore, the only hyper-parameter we have to set is the L_2 penalty `alpha` of the LSDA and DFDA models. For both cases we will select it using the `RandomizedSearchCV` model selection framework in `scikit-learn`. To select the optimal `alpha` values, we will perform 100 uniform random searches of `alpha` values in a range $(0, \alpha_{\max})$, averaging for each one its g scores over 10 cross validation folds built on the training set and retaining the value giving the largest validation score. The test g -scores are reported in Table 2 and the accuracies in Table 3. In each case the reported test g and accuracy values are obtained training 10 DLDA models with different random initializations, averaging their outputs as well as those of the corresponding test patterns and computing the class predictions and the test confusion matrix over these averages. The tables also give the rankings of each model over the different problems.

As it can be seen, the deep DFDA models clearly improve on the LSDA ones in terms of g scores and accuracies, with the best results usually obtained with the largest 5-hidden layer network. This is particularly remarkable on the

Table 2. Test g scores of the LSDA and of the DFDA models with 2, 3, 4 and 5 100 unit, hidden layers. We write in parenthesis the g score ranking of each model over the different problems and the corresponding ranking means in the last line.

Problem	LSDA	DFDA			
		2 HL	3 HL	4 HL	5 HL
combined	0.780 (5)	0.800 (3)	0.791 (4)	0.817 (2)	0.831 (1)
dna	0.940 (5)	0.952 (3)	0.952 (3)	0.955 (1)	0.953 (2)
ijcnn1	0.772 (5)	0.826 (3)	0.809 (4)	0.946 (2)	0.955 (1)
letter	0.683 (5)	0.924 (4)	0.936 (3)	0.952 (2)	0.957 (1)
mnist	0.870 (5)	0.929 (4)	0.947 (3)	0.952 (2)	0.962 (1)
pendigits	0.805 (5)	0.975 (4)	0.976 (3)	0.978 (2)	0.981 (1)
satimage	0.808 (5)	0.883 (2)	0.887 (1)	0.876 (3)	0.873 (4)
shuttle	0.565 (3)	0.000 (5)	0.414 (4)	0.838 (2)	0.979 (1)
w7a	0.794 (3)	0.694 (5)	0.776 (4)	0.852 (1)	0.849 (2)
w8a	0.781 (5)	0.837 (1)	0.827 (3)	0.827 (3)	0.831 (2)
usps	0.870 (4)	0.943 (2)	0.949 (1)	0.939 (3)	0.939 (3)
rank mean	4.55	3.27	3.00	2.09	1.73

Table 3. Test accuracies of the LSDA and of the DFDA models with 2, 3, 4 and 5 100 unit, hidden layers. Again, we write in parenthesis the accuracy ranking of each model over the different problems and the corresponding ranking means in the last line.

Problem	LSDA	DFDA			
		2 HL	3 HL	4 HL	5 HL
combined	0.770 (5)	0.791 (3)	0.782 (4)	0.808 (2)	0.819 (1)
dna	0.927 (5)	0.957 (3)	0.954 (4)	0.960 (1)	0.958 (2)
ijcnn1	0.855 (5)	0.891 (3)	0.883 (4)	0.977 (2)	0.984 (1)
letter	0.694 (5)	0.925 (4)	0.937 (3)	0.952 (2)	0.957 (1)
mnist	0.873 (5)	0.931 (4)	0.947 (3)	0.952 (2)	0.962 (1)
pendigits	0.825 (5)	0.975 (4)	0.976 (3)	0.978 (2)	0.981 (1)
satimage	0.835 (5)	0.891 (2)	0.897 (1)	0.885 (3)	0.882 (4)
shuttle	0.913 (5)	0.943 (3)	0.934 (4)	0.989 (1)	0.986 (2)
w7a	0.984 (3)	0.843 (5)	0.978 (4)	0.988 (2)	0.989 (1)
w8a	0.984 (5)	0.988 (2)	0.988 (2)	0.987 (4)	0.989 (1)
usps	0.883 (5)	0.949 (2)	0.954 (1)	0.946 (3)	0.944 (4)
rank mean	4.82	3.18	3.00	2.18	1.73

`shuttle` problem. Notice in Table 2 the g score of the 2-hidden layer network is 0, due to no pattern in the smallest class being correctly classified (this class has about 50,000 times less patterns than the biggest one).

5 Discussion and Further Work

While elegant and enticing, classical Fisher Linear Discriminant Analysis (FLDA) has fallen into some disuse, partly because of its linear nature but also because of the eigenanalysis it requires, which doesn't lend itself to be considered over very large datasets or to be learned in an iterative basis. Most of these difficulties are greatly alleviated when instead of a "pure" FLDA approach, one follows the equivalent LSR set-up proposed in [15, 16] and discussed above. Moreover, this lends itself into a natural extension to a Deep Neural Network setting, pairing the LSR target matrix with a highly complex deep pattern processing.

This is our approach here, where we have shown how simple 2-to-5 layer networks can noticeably improve the performance of FLDA. We have applied some of the latest tools in deep networks, such as Glorot initialization [7], RELU activations [11] or ADAM optimizers [10] but, in any case, the networks considered are relatively small and rather simple. There are thus several venues we can follow to improve on the results reported here. For instance, we can use other, more specialized, DNN architectures, easily available through the `keras` wrapper for Theano or TensorFlow, which may include convolutional layers for highly structured inputs such as `mnist`. These layers could also be helpful in problems such as person identification, where the current state of the art are the Fisher Face procedures, that apply a suitable version of Fisher analysis over face images. We could also consider DFDA networks as representation learners, using the network outputs of the last hidden layer as features upon which classifiers stronger than nearest neighbors or class mean distances could be applied. This may be particularly suitable for imbalanced problems. In this line it is also interesting to compare the performance of Deep Fisher networks with that of other non linear Fisher extensions, such as KDA. We are currently studying these issues.

Acknowledgments. With partial support from Spain's grants TIN2013-42351-P, TIN2016-76406-P, TIN2015-70308-REDT and S2013/ICE-2845 CASI-CAM-CM. Work supported also by project FACIL-Ayudas Fundación BBVA a Equipos de Investigación Científica 2016, the UAM-ADIC Chair for Data Science and Machine Learning and Instituto de Ingeniería del Conocimiento. The third author is also supported by the FPU-MEC grant AP-2012-5163. We gratefully acknowledge the use of the facilities of Centro de Computación Científica (CCC) at UAM.

References

1. Belhumeur, P.N., Hespanha, J.P., Kriegman, D.J.: Eigenfaces vs. fisherfaces: recognition using class specific linear projection. *IEEE Trans. Pattern Anal. Mach. Intell.* **19**(7), 711–720 (1997). <http://dx.doi.org/10.1109/34.598228>
2. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. In: *Proceedings of Python for Scientific Computing Conference (SciPy)*, Oral Presentation, Jun 2010
3. Chollet, F.: Keras: deep learning library for Theano and TensorFlow (2015). <https://github.com/fchollet/keras>

4. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edn. Wiley, Hoboken (2000)
5. Friedman, J.H.: Regularized discriminant analysis. *J. Am. Stat. Assoc.* **84**(405), 165–175 (1989)
6. Fukunaga, K.: Introduction to Statistical Pattern Recognition. Computer Science and Scientific Computing. Academic Press, Boston (1990)
7. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: *JMLR W&CP: Proceedings of 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, pp. 249–256, May 2010
8. González, A.M., Dorronsoro, J.R.: Natural learning in NLDA networks. *Neural Netw.* **20**(5), 610–620 (2007). <http://dx.doi.org/10.1016/j.neunet.2006.09.014>
9. Google: Tensorflow, an open source software library for machine intelligence. <https://www.tensorflow.org/>
10. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. *CoRR abs/1412.6980* (2014). <http://arxiv.org/abs/1412.6980>
11. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105. Curran Associates, Inc. (2012). <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
12. Mika, S., Rätsch, G., Weston, J., Schölkopf, B., Smola, A.J., Müller, K.: Invariant feature extraction and classification in kernel spaces. In: *Advances in Neural Information Processing Systems, NIPS Conference, Denver, Colorado, USA, 29 November – 4 December 1999*, vol. 12, pp. 526–532 (1999)
13. Park, C.H., Park, H.: A relationship between linear discriminant analysis and the generalized minimum squared error solution. *SIAM J. Matrix Anal. Appl.* **27**(2), 474–492 (2005). <http://dx.doi.org/10.1137/040607599>
14. Santa Cruz, C., Dorronsoro, J.R.: A nonlinear discriminant algorithm for feature extraction and data classification. *IEEE Trans. Neural Netw.* **9**(6), 1370–1376 (1998). <http://dx.doi.org/10.1109/72.728388>
15. Ye, J.: Least squares linear discriminant analysis. In: *Proceedings of 24th International Conference on Machine Learning (ICML 2007)*, Corvallis, Oregon, USA, 20–24 June 2007, pp. 1087–1093 (2007)
16. Zhang, Z., Dai, G., Xu, C., Jordan, M.I.: Regularized discriminant analysis, ridge regression and beyond. *J. Mach. Learn. Res.* **11**, 2199–2228 (2010)

Deep MLPs for Imbalanced Classification

David Díaz–Vico

Instituto Ingeniería del Conocimiento
Universidad Autónoma de Madrid

Madrid, Spain
david.diazv@iic.uam.es

Aníbal R. Figueiras–Vidal

Signal Th. & Communications Department
Universidad Carlos III

Madrid, Spain
arf@tsc.uc3m.es

José R. Dorronsoro

Dept. Computer Engineering,
Instituto Ingeniería del Conocimiento
Universidad Autónoma de Madrid

Madrid, Spain
jose.dorronsoro@uam.es

Abstract—Classification over imbalanced datasets is a highly interesting topic given that many real-world classification problems present a concrete class with a much smaller number of patterns than the others. In this work we shall explore the use of large, fully connected and potentially deep MLPs in such problems. We will consider simple MLPs, with ReLU activations, softmax outputs and categorical cross-entropy loss, showing that, when properly regularized, these relatively straightforward MLP models yield state of the art results in terms of the areas under the ROC curve for both two-class problems (the usual focus in imbalanced classification) as well as for multi-class problems.

I. INTRODUCTION

Imbalanced classification is a very important problem in Machine Learning which has received enormous attention, particularly since the mid 2000's, when there was an explosion on the published research [1], [2]. Usually two main general venues are followed for imbalanced classification, sample-based and algorithm-based [3]. In the first approach one tries to correct the initial sample imbalance by either oversampling the smaller class [4], undersampling the largest one [5] or combining both; often this is done in an ensemble setting [6] such as building independent ensemble members after rebalancing the data or within an adaptive boosting set-up procedure. Because of the computational cost of the rebalancing involved, it is often the case that simple classifiers such as decision trees are used. Stronger classifiers are usually applied in algorithm-based methods, whose general goal is to correct the bias that they may have towards the larger classes; this may be done either by introducing classification costs adapted to the imbalance structure at hand [7], or by modifying the loss function to give a greater impact to the smaller classes [8].

An underlying assumption here is that the smaller classes will not be “noticed” by the classifier. A naive way to avoid this would be to work with overfitting classifiers which, at least during training, would be able to correctly handle the minority classes; the danger is, of course, a potentially very poor generalization. Regularization techniques are used to correct this but, if not carefully chosen, there is a risk of the minority classes being lost again. One way out of this could be to devise specific regularization penalties to counter it; another, simpler one is to use standard regularizers but to choose their penalty parameter in such a way that the minority classes still count.

In general, a desirable characteristic of any classifier is its ability to provide continuous scores such as, for instance,

posterior probability estimates. Classification decisions are often made by setting a threshold but its choice is always tricky and more so in imbalanced problems, where there is little or none a priori information to do so. In fact in, say, a two-class imbalanced problem, any score value is bound to be represented by patterns with a substantial overlap between the positive and negative classes. Because of this, single valued 0–1 classification performance measures based on confusion matrix values computed over a fixed score threshold, such as accuracy, precision or recall, may not be too helpful and a more nuanced measure which takes into account all possible thresholds should be considered.

The most obvious such choice is the Area Under the Curve (*AUC*) of the Receiver Operating Characteristics (*ROC*) [9]. For two classes, *ROC* measures true positive rates (TPR) against false positive rates (FPR) over all possible thresholds and, in fact, the *AUC* estimates the probability of a given positive pattern receiving a positive score larger than a negative one (see Section II for details). The *AUC* may not be only used to measure a classifier's performance but also to guide the selection of its optimal hyper-parameters such as, for instance, those involved in regularization. In fact, optimal hyper-parameters chosen as to maximize the *AUC* on a validation subset should also be effective when the classifier is applied to a test set.

In this work we shall try to exploit these two insights, choosing first Multilayer Perceptrons (MLPs) with soft-max outputs and cross-entropy loss; their output can be naturally taken as posterior probability estimates and, hence, as a suitable score. With some abuse of the language, we will refer to our MLPs as “Deep” MLPs; while the maximum number of layers we will consider will be five (and, hence, far from the much larger layer numbers in use nowadays), our networks will certainly have a large number of weights, well above sample size in many of the problems we will consider. As they are defined with architectures large enough this will result in powerful non-linear classifiers, susceptible to produce overfitting if not properly handled. To prevent this, we will use our second insight of using the *AUC* as the criterion function in hyper-parameterization.

In principle one might consider many different hyper-parameters when defining a given MLP, such as architectures or learning rates. However the continuous recent advances in Deep Neural Networks and the better understanding of MLP

training that they have produced suggest that large MLPs can be rather robust with respect to choices such as the number of hidden layers and their number of units. Moreover, the latest stochastic gradient descent–based optimization techniques such as Adam [10] do automatically provide curvature–adapted learning rates and yield good results when their default hyper–parameters are used. This essentially leaves the regularization parameter as the key value to be adjusted in order to have good generalization performance. Here we will use a standard L_2 or Tikhonov regularization term $\frac{\alpha}{2}\|W\|_2^2$, with W the weight set of the MLP and, as pointed out before, the optimal α will be chosen by cross-validation using the AUC score.

This is the set up we will follow here and besides a short, self–contained presentation of some relevant facts on the AUC and deep network training, our contributions are the following:

- The proposal of an AUC based procedure to select the optimal L_2 regularization parameter α for log–loss, softmax deep MLP multi–class classifiers.
- An extensive study of the performance of log–loss, softmax MLP classifiers on two class imbalanced problems, showing results that improve in many cases on the current state of the art methods.
- The application of these MLPs to large scale, imbalanced multiclass problems, an area where research on imbalanced problems has been somewhat less active, and where we can also obtain quite remarkable results

In summary, we show that modern deep MLP techniques can easily and naturally be applied to imbalanced multiclass problems in an almost routine fashion and that these more or less straightforward MLPs can yield state of the art results.

The paper is organized as follows. In Section II we shall review the most relevant facts about AUC and its multi–class extensions and in Section III we will briefly review logloss, softmax deep MLP training. These MLPs are compared in Section IV with state of the art methods for two–class imbalanced problems and applied in Section V to large scale, imbalanced multiclass problems. The paper ends with a brief discussion and pointers to further work.

II. THE AREA UNDER THE ROC CURVE FOR TWO– AND MULTI–CLASS PROBLEMS

A. Two–class Problems

Assume we have a 2–class discriminant that for each element x returns a score $s(x)$ which ranks higher elements of the positive (1) class, such as, for instance, a posterior probability estimate $\hat{P}(1|x)$; for simplicity we will assume s normalized so that $0 \leq s(x) \leq 1$. Let S_0, S_1 be the random variables defined by s on the 0 and 1 classes, that we assume to be independent with densities f_0, f_1 . For any threshold t consider the classifier $c(x) = c_t(x)$ such that $c(x) = 1$ iff

$s(x) > t$. If $TPR(t), FPR(t)$ are its true and false positive ratios we have

$$\begin{aligned} TPR(t) &= P(\{s(x) > t | x \in C_1\}) = \int_t^1 f_1(s)ds, \\ FPR(t) &= P(\{s(x) > t | x \in C_0\}) = \int_t^1 f_0(s)ds. \end{aligned}$$

The points $(FPR(t), TPR(t))$ lie in the Receiver Operating Characteristics (ROC) space. For instance, perfect classification would result on the $(0, 1)$ point while random classification would result in a point on the $y = x$ diagonal. Notice that TPR is equivalent to recall or sensitivity and FPR is equal to specificity. and sometimes the x and y ROC axes are labeled as specificity and sensitivity. Letting t vary, these points define the ROC curve and the AUC (i.e. Area Under the Curve), is simply the area underlying it.

The AUC has a number of interesting properties. First, no prior probabilities enter its definition, as $TPR(t)$ and $FPR(t)$ are computed column–wise on the confusion matrix. Moreover, the AUC captures in a single metric classifier performance on a wide range of decision thresholds. Also, it gives the probability of a random positive pattern being ranked higher than a random negative one (see for instance [11]). In fact, we have

$$\begin{aligned} AUC &= \int_0^1 TPR(t)d(-FPR(t)) \\ &= \int_0^1 \left(\int_t^1 s f_1(s)ds \right) f_0(t)dt \\ &= \int_0^1 \int_0^1 I(s > t) f_0(t) f_1(s) dt ds \\ &= P(\{s(x) > s(x') : x \in C_1, x' \in C_0\}). \end{aligned}$$

Notice that $FPR(t)$ decreases with t , which requires working with $d(-FPR(t))$. We have defined the AUC in terms of the 1–class (i.e., the AUC_1), but we get an equivalent value using the true and false negative rates $TNR(t)$ and $FNR(t)$. In fact,

$$(FNR(t), TNR(t)) = (1 - TPR(t), 1 - FPR(t)),$$

i.e., the 0–class ROC curve is symmetric to the 1–class curve with respect the line $y = 1 - x$, and their corresponding AUC s thus verify $AUC_0 = AUC_1$.

The AUC is also related to the Mann–Whitney U rank test, as proved in [12]. In fact, let’s sort the sample scores as s_n , label the patterns x accordingly as x_n and define

$$U_1 = \sum_{i=1}^{N_1} r_i^1 - \frac{N_1(N_1 + 1)}{2} = \sum_{n=1}^N n I_1(x_n) - \frac{N_1(N_1 + 1)}{2},$$

where r_i^1 is the rank in the overall sorting of the score s of the i –th element of class 1 and $I_1(x_n)$ the 0–1 indicator of class 1; we then have $AUC_1 = \frac{1}{N_0 N_1} U_1$. This can be easily seen from the fact that

$$AUC = P(\{s(x_j^0) < s(x_i^1) : x_j^0 \in C_0, x_i^1 \in C_1\}).$$

In fact, the number of (x_j^0, x_i^1) pairs is N_0N_1 and if we assume the elements x_i^1 of C_1 ordered according to ascending s scores, the number of elements x for which $s(x) < s(x_i^1)$ is $r_i - 1$, while the number of such elements in C_0 is $i - 1$. We thus have

$$\begin{aligned} AUC &= \frac{1}{N_0N_1} \sum_{i=1}^{N_1} \#\{x \in C_0 : s(x) < s(x_i^1)\} \\ &= \frac{1}{N_0N_1} \sum_{i=1}^{N_1} (r_i^1 - 1 - (i - 1)) \\ &= \frac{1}{N_0N_1} \sum_{i=1}^{N_1} r_i^1 - \frac{(N_1 + 1)}{2N_0} \end{aligned} \quad (1)$$

thus, AUC can be directly derived from the Mann–Whitney value U_1 or we can simply use (1) to compute it.

B. Multi-class Problems

The extension of the AUC to a multiclass setting is more complex. Closest to the previous two class is the Volume Under the Surface (VUS) approach [13] that, however, may be quite hard to compute for score-based classifiers. Most simplifications derive multiclass AUC starting from various two class AUC values and combining them somehow. For instance, in [14] a two class problem is considered for each class C_i , with is taken as the positive class, and patterns in the other classes $\cup_{j \neq i} C_j$ as the negative one, the corresponding AUC_i is computed and the **total** AUC is taken as

$$AUC_{total} = \sum_i \pi_i AUC_i,$$

with π_i the class priors. Notice that this definition may be quite sensible to class imbalances. An alternative to AUC_{total} are the so-called **macro-average** AUC s [15], defined as either the arithmetic mean of the previous one-versus-rest (o-v-r) AUC_i values:

$$AUC_{macro} = \frac{1}{C} \sum_i AUC_i,$$

or by its geometric variant

$$AUC_{geom} = \left(\prod_1^C AUC_i \right)^{\frac{1}{C}}.$$

A single ROC version of macro-averaging can be found on the ROC Multiclass Settings (scikit-learn.org/stable/auto_examples/model_selection/plot_roc) discussion on Scikit-learn’s page. In it, once the TPR_i and FPR_i vectors are computed for each class i and the entire threshold range R , these FPR_i are concatenated into a single vector FPR with $C \times \text{length}(R)$ values and a mean TPR vector is obtained by averaging the interpolations of the pairs FPR_i, TPR_i over the FPR values. This resulting FPR and TPR vectors can be used to build a global, macro-averaged ROC curve and the final AUC_{macro} is just the area of this global curve.

Still, another alternative is to **micro-average** AUC s [16], Ch. 13, by raveling the target and score matrices to build a unique pair of $(target, score)$ vectors and compute from this pair the so-called micro-average ROC and its corresponding overall AUC_{micro} . Notice that for any threshold t the total numbers of true positives and false negatives in the raveled target and score vectors are

$$TP(t) = \sum_1^C TPI_i(t), \quad FN(t) = \sum_1^C FNI_i(t)$$

while the total number of positives is $\sum_1^C P_i$; we can thus compute this way the micro-averaged $TPR(t)$ and $FPR(t)$ ratios. Finally, another widely used alternative to computing a multi-class AUC -like value is the so-called M - AUC [12], where $C(C - 1)$ pairwise two class $AUC(i, j)$ values are computed and merged into

$$M = \frac{2}{C(C - 1)} \sum_{i < j} AUC(c_i, c_j),$$

with each $AUC(c_i, c_j)$ considering the effect of the score s only on patterns of classes i and j . This definition should be more robust against class imbalances but, in any case, most of the probabilistic interpretations of the two class AUC are lost for both AUC_{total} definitions. See Section 9 in [11] for more details. Examples of the use of micro and macro averaging can be found for instance in [15], [17]. While there is no general agreement on which measure to use over unbalanced datasets, macro-averaging seems to be often used in the multi-class setting. We shall use geometric macro AUC in our experiments.

C. Consistency and Discriminancy

As it is outlined in [18], the degrees of consistency and discriminancy of two measures provide an objective comparison between them. More precisely, in [18] the following two definitions are introduced.

Definition 2.1: For two measures f and g on a domain Ψ , let $R = \{(a, b) \mid a, b \in \Psi, f(a) > f(b), g(a) > g(b)\}$ and $S = \{(a, b) \mid a, b \in \Psi, f(a) > f(b), g(a) < g(b)\}$. The **degree of consistency** of f and g is $C(0 \leq C \leq 1)$, where $C = \frac{|R|}{|R| + |S|}$.

Definition 2.2: For two measures f and g on a domain Ψ , let $P = \{(a, b) \mid a, b \in \Psi, f(a) > f(b), g(a) = g(b)\}$, $Q = \{(a, b) \mid a, b \in \Psi, g(a) > g(b), f(a) = f(b)\}$. The **degree of discriminancy** for f over g is $D = \frac{|P|}{|Q|}$.

Consider now ranked lists of the form $L = \{s_n, \ell_n\}$, $1 \leq n \leq N$, with $0 \leq s_n \leq 1$ a binary score and $\ell_n = 0, 1$ a binary class label. We can compute the accuracy over L as simply $acc(L) = \frac{|\{s_n > 0.5, \ell_n = 1\}|}{N}$ and $AUC(L)$ according to (1). Then, it is shown in [18], Theorem 1, that, in the above notation, $|R| > |S|$, i.e., $C > 0.5$ or, in the terminology of [18], that AUC is more consistent than accuracy. Moreover, if we assume the L to be balanced, i.e., to have the same number of positive and negative patterns, it is also proved in Theorem 2 of [18] that $|P| > |Q|$, i.e., $D > 1$ or, in the terminology

of [18], that AUC is more discriminative than accuracy. Although the latter result is not proved for imbalanced lists, in the experiments in [18] on synthetic and real-world datasets, AUC also turns out to be more discriminative in imbalanced problems. In light of these facts, only AUC will be used as the default metric in the experimental section of this study.

III. DEEP MLPs FOR CLASSIFICATION

In our experiments we will work with fairly straightforward fully connected Multilayer Perceptrons (MLPs) which we briefly review next. As it is usually done, we will label the targets by 1-hot encoding, i.e., with $e^c = (0, \dots, \underbrace{1}_c, \dots, 0)$

being the target for class c elements. Then if $x \in \mathcal{C}_c$, we can write $P(c|x) = \prod_{k=1}^K P(k|x)^{e_k^c}$. Therefore, for a posterior probability $P(c|x, W)$, the probability of getting a sample $S = \{x^p, y^p = e^{c(p)}\}$ is

$$P(S; W) = \prod_1^N P(c(p)|x^p; W) = \prod_1^N \prod_{k=1}^K P(k|x^p; W)^{e_k^{c(p)}}$$

and its log-likelihood becomes

$$\begin{aligned} \ell(W; S) &= \log P(S; W) \\ &= \sum_{p=1}^N \sum_{k=1}^K \log \left(P(k|x^p; W)^{e_k^{c(p)}} \right) \\ &= \sum_{p=1}^N \sum_{k=1}^K y_k^p \log P(k|x^p; W). \end{aligned}$$

We will estimate the optimal weights W^* by minimizing $-\ell(W; S)$, i.e., the log loss

$$L_{\log}(W) = - \sum_{p=1}^N \sum_{k=1}^K y_k^p \log P(k|x^p; W)$$

to which we will add an L_2 -Tikhonov regularizer $\frac{\alpha}{2} \|W\|_2^2$. The general, classic theory of such networks is very well known (see for instance [19], Chapter 5) but the recent developments on deep NNs have resulted in a number of important enhancements, some of which we will apply here. For instance, we will use ReLUs [20] as the activation functions, Glorot-Bengio weight initialization [21] and minibatch-based Adam [10] as the optimization method. All these options are available in neural network frameworks such as Keras [22] or Scikit-learn [23]. Here we will use Scikit-learn's `MLPClassifier` class. Notice that in a strict sense, our networks are not very deep, as we consider at most five hidden layers. In any case, we will refer to them as "deep" as we will use the recent advances on deep networks that have made possible the relatively easy training of networks with several layers and a very large number of weights (which in our case can be in the tens of thousands).

To minimize the computational effort in hyper-parameter estimation, we have considered fixed MLP architectures with 0 to 5 hidden layers, each with 100 ReLU units and softmax

Table I
CHARACTERISTICS OF THE SMALL DATASETS.

Dataset	dimension	n. patterns	class ratio
abalone9-18	8	731	16.4
abalone19	8	4174	129.4
ecoli4	7	336	15.8
ecoli-0-1-3-7_vs_2-6	7	281	39.14
ecoli-0-1-4-6_vs_5	6	280	13
ecoli-0-1-4-7_vs_2-3-5-6	7	336	10.6
ecoli-0-1-4-7_vs_5-6	6	332	12.3
ecoli-0-1_vs_5	6	240	11.0
ecoli-0-3-4-6_vs_5	7	205	9.3
ecoli-0-3-4-7_vs_5-6	7	257	9.3
ecoli-0-6-7_vs_5	6	220	10
glass2	9	214	11.6
glass4	9	214	15.5
glass5	9	214	22.8
glass-0-1-4-6_vs_2	9	205	11.1
glass-0-1-6_vs_2	9	192	10.3
glass-0-1-6_vs_5	9	184	19.4
glass-0-4_vs_5	9	92	9.2
glass-0-6_vs_5	9	108	11
page-blocks-1-3_vs_4	10	472	15.9
shuttle-c0-vs-c4	9	1829	13.9
shuttle-c2-vs-c4	9	129	20.5
vowel0	13	988	10.0
yeast4	8	1484	28.1
yeast5	8	1484	32.7
yeast6	8	1484	41.4
yeast-0-5-6-7-9_vs_4	8	528	9.4
yeast-1-2-8-9_vs_7	8	947	30.6
yeast-1-4-5-8_vs_7	8	693	22.1
yeast-2_vs_8	8	482	23.1

activations in the output layer (the use of asymmetric final activations for imbalanced problems has been recently suggested in [24]). We have also used the default hyper-parameters for the Adam solver. This leaves the L_2 penalty parameter `alpha` as the only one to hyper-parameterize, which we do by 10-fold cross-validation considering a logarithmic equi-spaced grid with values $\alpha = 2^q$ with $-30 \leq q \leq 10$.

IV. DEEP MLPs IN IMBALANCED TWO-CLASS PROBLEMS

In this section we shall consider the application of deep MLPs on a series of two-class imbalanced problems obtained from the Keel repository [25]; we will use their 5-fold train-test splits (just as it is done in [26]). Their size, dimension and class imbalance ratio are given in Table I; the numbers in the dataset names indicate either which class is selected in a one-versus-rest scenario (as in `abalone19`) or the several classes selected in some-versus-others scenario (as in `ecoli-0-1-3-7_vs_2-6`).

We shall consider MLPs with 0 to 5 hidden layers with 100 units each, which are regularized using the squared L_2 norm of the entire weight set (notice that for 0 layers this essentially reduces to Logistic Regression). For each problem, and for each of its 5 folds, we shall perform a hyper-parameter grid search over the train subset to select the optimal α regularization parameter by stratified 5-fold cross-validation (CV). The scoring used is the AUC of the positive class (as mentioned, $AUC_1 = AUC_0$). Once the optimal α is chosen,

an individual MLP is trained on the complete train subset and the probability scores and *AUC*s are computed on the test subset; finally we average these values over the 5 folds.

Table II gives the *AUC* test results for the MLP classifiers over the datasets considered (for space reasons we omit those of MLPClassifier1, which are similar to the others). For easier analysis the table also contains the rankings of the five MLPs considered according to their test *AUC* values. As it can be seen, the results for all MLPs are quite similar. In fact, the bottom line of the table gives the average rank for the *AUC* values of each classifier and the difference between the average rank of the 0-layer MLP (which is equivalent to a logistic regression model) and that of the 5-layer MLP is very small. In order to provide an objective comparison between the models, we have applied Wilcoxon paired signed-rank tests over the *AUC* columns of the different MLPs. Its results are shown in table III and the null hypothesis of same means cannot be rejected in any case. Probably these similar results are due to the relative simplicity of the datasets involved.

For comparison purposes, Table II also contains the best test *AUC* values reported in [26] for the test subsets computed using six procedures USwitchingNED, EUSB, UBAG, SBAG, SBO, EASY, representative of the current state of the art. As it can be seen in the table, the *AUC* values provided by the MLPs are higher in almost all cases but it must be pointed out that this comparison is to be taken in its proper terms for the goals and, most importantly, the underlying classifiers (relatively weak decision trees versus quite powerful deep networks) are quite different here from those in [26].

V. DEEP MLPs IN LARGE SCALE IMBALANCED MULTI-CLASS PROBLEMS

Although having also a clear importance in applications, classification over large-scale imbalanced multi-class datasets seems to have received much less attention in the literature [27]. We will now consider some moderately bigger, multi-class datasets available on the datasets section of the LIBSVM web site. Some of them (namely, *dna*, *ijcnn1*, *letter*, *satimage*, *shuttle*) have separate pre-defined train-validation-test splits which we will use to determine the optimal value of the regularization parameter α . For the other (namely, *a4a*, *a8a*, *cod-rna*, *combined*, *news20*, *pendigits*, *usps*, *w7a*, *w8a*), this parameter is estimated by 10-fold cross validation on the train subset. In Table IV we give their dimension, total number of train patterns, number of classes and their maximum class size ratios, i.e., the ratio of the maximum class size to the minimum one. Data sizes go from 2,000 (*dna*) to 78,823 (*combined*) and the number of classes ranges from 2 to 26 (*letter*); while some of them are quite balanced, others (*w7a*, *w8a* and particularly *shuttle*) present large class imbalances.

Again, we consider MLPs with 0 to 5 hidden layers with 100 units each, which are regularized using the L_2 of the entire weight set. In this case, the hyper-parameter search is performed using the train-validation partition, if it is available, or with 10-fold over the train subset if it is not. Once the optimal hyper-parameter is selected, the train (and validation if

exists) subset is again used to train the final model. Finally, the multiclass geometric *AUC* is computed over the test subset with the results displayed in table V, which also gives the ranking of each MLP for an easier comparison.

While not always true, it seems the deeper the model, the better the *AUC* reported, though differences between the different models are relatively small in most cases. Table VI shows the results of applying the paired Wilcoxon signed rank tests to the Table's columns (we point out that the smaller number of *AUC* values on the columns may make problematic the usual distribution approximations made in Wilcoxon's test). In any case, the table suggests that there are three categories, that of MLPClassifier0, that of MLPClassifier2 and MLPClassifier3 and, finally, that of MLPClassifier4 and MLPClassifier5. Of these, MLPClassifier0 seems to be worse than all the others and MLPClassifier2 and MLPClassifier3 also seem to have a worse performance than MLPClassifier4 and MLPClassifier5. On the other hand, the MLPClassifier2-MLPClassifier3 and MLPClassifier4-MLPClassifier5 pairs seem to perform similarly. This can also be seen in the average rank values at the bottom line of Table VI.

VI. DISCUSSION AND CONCLUSIONS

Imbalanced classification problems are usually studied through either sample-based methods that somehow try to rebalance the initial samples or algorithm-based methods which try to give a bigger relevance to the minority class. These techniques define the current state-of-the-art but the sample based methods require subsampling techniques that are rather delicate to implement and, as they require repeated subsampling and training cycles, often are built on top of relatively simple base classifiers. On its part, algorithm-based methods often need classification costs which are quite dependent on the particular sample being studied and which may not be adequate for a different problem. On the other hand, the huge attention that deep neural networks have received in the past few years has resulted in quite robust training techniques that ensure convergence even when the number of weights is much bigger than sample size. Moreover, when coupled with drop-out regularization, the cross-entropy loss may be particularly well suited to build deep networks able to achieve near optimal representations, as described in [28].

Here we have followed a much simpler path, working with log-loss, softmax MLP classifiers and L_2 regularization, using *AUC* as the criterion function to direct the selection of the optimal L_2 regularization parameter. Our MLP architectures are fairly simple, just involving a number of fully connected feed-forward layers but our experimental results indicate that the resulting MLPs are competitive with the current state of the art, even in highly imbalanced problems. Moreover, the cross-entropy loss easily extends to multiclass setting without the need to consider multiple one-vs-the rest problems and, finally, these MLPs can be applied to large scale problems with a computational cost that scales better than that of other non-linear classifiers.

Table II
MAXIMUM AUC VALUES IN [26] AND TEST AUC FOR THE 0– AND 2– TO 5–HIDDEN LAYER MLPs CONSIDERED.

	U/OS [26]	MLPClassifier0	MLPClassifier2	MLPClassifier3	MLPClassifier4	MLPClassifier5
abalone9-18	72.94	95.85 ± 0.98 (4)	96.19 ± 2.43 (2)	96.50 ± 2.52 (1)	95.69 ± 2.84 (5)	95.89 ± 2.75 (3)
abalone19	69.91	81.78 ± 3.39 (2)	81.43 ± 5.12 (4)	81.98 ± 5.20 (1)	80.90 ± 5.27 (5)	81.46 ± 5.03 (3)
ecoli4	92.15	99.68 ± 0.39 (1)	99.45 ± 0.59 (4.5)	99.45 ± 0.59 (4.5)	99.60 ± 0.61 (2)	99.53 ± 0.30 (3)
ecoli-0-1-3-7_vs_2-6	83.54	98.36 ± 2.84 (2)	97.08 ± 4.24 (5)	97.45 ± 4.24 (3)	98.72 ± 1.59 (1)	97.08 ± 4.96 (4)
ecoli-0-1-4-6_vs_5	90.96	93.56 ± 7.89 (2)	93.75 ± 8.44 (1)	93.17 ± 8.19 (3)	93.12 ± 10.04 (4)	92.60 ± 9.17 (5)
ecoli-0-1-4-7_vs_2-3-5-6	88.72	94.42 ± 4.94 (1)	93.54 ± 5.68 (2)	93.01 ± 5.89 (3)	92.64 ± 6.41 (4)	92.29 ± 5.52 (5)
ecoli-0-1-4-7_vs_5-6	90.96	94.96 ± 5.18 (2)	95.35 ± 5.29 (1)	94.45 ± 6.87 (4)	94.77 ± 6.84 (3)	94.29 ± 7.22 (5)
ecoli-0-1_vs_5	91.14	91.59 ± 8.05 (5)	92.95 ± 9.27 (2)	92.95 ± 8.60 (2)	92.84 ± 8.57 (4)	92.95 ± 8.60 (2)
ecoli-0-3-4-6_vs_5	91.33	92.97 ± 6.66 (1)	92.16 ± 8.11 (2.5)	91.62 ± 8.07 (4)	91.22 ± 8.32 (5)	92.16 ± 8.11 (2.5)
ecoli-0-3-4-7_vs_5-6	90.76	94.62 ± 7.95 (3)	94.89 ± 9.79 (1)	94.12 ± 10.69 (5)	94.46 ± 9.64 (4)	94.72 ± 9.70 (2)
ecoli-0-6-7_vs_5	90.25	91.75 ± 8.38 (2)	92.50 ± 8.74 (1)	88.50 ± 9.46 (5)	89.47 ± 9.60 (3)	88.65 ± 9.70 (4)
glass2	74.75	82.48 ± 15.70 (5)	89.97 ± 7.66 (4)	90.28 ± 7.27 (2)	90.11 ± 7.28 (3)	91.18 ± 5.85 (1)
glass4	90.01	95.35 ± 3.71 (3)	94.64 ± 4.71 (5)	94.88 ± 5.45 (4)	96.61 ± 3.07 (2)	97.09 ± 3.27 (1)
glass5	98.78	99.27 ± 1.46 (3)	99.02 ± 1.95 (5)	99.27 ± 1.46 (3)	99.76 ± 0.49 (1)	99.27 ± 1.46 (3)
glass-0-1-4-6_vs_2	78.78	79.17 ± 10.32 (5)	85.76 ± 7.52 (4)	87.36 ± 8.08 (1)	86.11 ± 8.21 (3)	86.96 ± 7.22 (5)
glass-0-1-6_vs_2	70.03	82.00 ± 9.38 (5)	86.29 ± 6.01 (1)	86.24 ± 9.60 (2)	83.82 ± 7.65 (4)	84.19 ± 7.39 (3)
glass-0-1-6_vs_5	97.14	99.71 ± 0.57 (1)	97.14 ± 4.43 (5)	98.86 ± 1.40 (3.5)	98.86 ± 1.07 (3.5)	99.14 ± 1.14 (2)
glass-0-4_vs_5	99.41	97.50 ± 3.06 (5)	98.75 ± 2.50 (4)	100.00 ± 0.00 (2)	100.00 ± 0.00 (2)	100.00 ± 0.00 (2)
glass-0-6_vs_5	99.50	100.00 ± 0.00 (3)	100.00 ± 0.00 (3)	100.00 ± 0.00 (3)	100.00 ± 0.00 (3)	100.00 ± 0.00 (3)
page-blocks-1-3_vs_4	99.78	97.39 ± 1.20 (5)	99.93 ± 0.15 (4)	100.00 ± 0.00 (2)	100.00 ± 0.00 (2)	100.00 ± 0.00 (2)
shuttle-c0-vs-c4	100.0	99.20 ± 1.60 (3)	99.20 ± 1.60 (3)	99.20 ± 1.60 (3)	99.20 ± 1.60 (3)	99.20 ± 1.60 (3)
shuttle-c2-vs-c4	100.0	99.20 ± 1.60 (4.5)	99.20 ± 1.60 (4.5)	100.00 ± 0.00 (2)	100.00 ± 0.00 (2)	100.00 ± 0.00 (2)
vowel0	98.26	99.37 ± 0.48 (5)	100.00 ± 0.00 (2.5)	100.00 ± 0.00 (2.5)	100.00 ± 0.00 (2.5)	100.00 ± 0.00 (2.5)
yeast4	84.49	88.51 ± 4.19 (1)	87.90 ± 5.75 (2)	87.06 ± 6.23 (3)	86.86 ± 5.05 (4)	86.82 ± 4.95 (5)
yeast5	97.05	98.77 ± 0.18 (5)	98.89 ± 0.31 (4)	98.90 ± 0.38 (3)	98.97 ± 0.20 (2)	99.03 ± 0.32 (1)
yeast6	88.08	94.04 ± 6.11 (1)	93.82 ± 6.64 (4)	93.15 ± 6.61 (5)	93.89 ± 6.97 (2)	93.86 ± 6.06 (3)
yeast-0-5-6-7-9_vs_4	80.46	83.65 ± 4.72 (5)	86.99 ± 6.32 (4)	87.13 ± 6.49 (3)	87.82 ± 7.70 (2)	87.93 ± 3.72 (1)
yeast-1-2-8-9_vs_7	70.20	81.76 ± 2.68 (1)	80.29 ± 5.79 (2)	78.14 ± 9.67 (5)	78.69 ± 4.58 (4)	79.18 ± 5.89 (3)
yeast-1-4-5-8_vs_7	59.18	70.22 ± 6.34 (5)	71.39 ± 7.05 (3)	72.60 ± 5.40 (1)	72.59 ± 12.67 (2)	70.47 ± 11.68 (4)
yeast-2_vs_8	78.67	85.30 ± 4.93 (1)	83.94 ± 4.92 (2)	77.16 ± 14.01 (5)	83.87 ± 6.66 (3)	79.62 ± 10.15 (4)
rank mean		3.1	3.1	3.0	3.0	2.9

Table III
RESULTS OF THE WILCOXON TESTS FOR TWO–CLASS PROBLEMS.

	p-value	Hypothesis
MLPClassifier0 vs MLPClassifier2	0.37	Not rejected
MLPClassifier0 vs MLPClassifier3	0.79	Not rejected
MLPClassifier0 vs MLPClassifier4	0.37	Not rejected
MLPClassifier0 vs MLPClassifier5	0.72	Not rejected
MLPClassifier2 vs MLPClassifier3	0.80	Not rejected
MLPClassifier2 vs MLPClassifier4	0.96	Not rejected
MLPClassifier2 vs MLPClassifier5	0.74	Not rejected
MLPClassifier3 vs MLPClassifier4	0.50	Not rejected
MLPClassifier3 vs MLPClassifier5	0.61	Not rejected
MLPClassifier4 vs MLPClassifier5	0.93	Not rejected

In a broad sense, classification methods will be more successful if they may produce new representations of the input data that increase class separation while preserving interclass distances. Recent studies [29] point out how deep networks may help in that direction; if so, they would certainly be rather effective in imbalanced problems. These observations and the use of better tailored regularizers such as the just mentioned dropout or other related to the information bottleneck loss [28] offer venues for further research.

Acknowledgments.: With partial support from Spain’s grants TIN2013-42351-P, TIN2016-76406-P, TIN2015-70308-REDT and S2013/ICE-2845 CASI-CAM-CM. Work supported also by project FACIL–Ayudas Fundación BBVA a Equipos de

Table IV
TRAIN SAMPLE SIZE, DIMENSION, NUMBER OF CLASSES AND MAXIMUM TO MINIMUM CLASS SIZE RATIO FOR THE MULTI–CLASS DATASETS.

Problem	n. patterns	dimension	n. classes	class ratios
a4a	4,781	123	2	3.0
a8a	22,696	123	2	3.1
combined	78,823	100	3	2.2
dna	2,000	180	3	2.3
ijcnn1	49,990	22	2	9.3
letter	15,000	16	26	1.1
pendigits	7,494	16	10	1.1
satimage	4,435	36	6	2.6
shuttle	43,500	9	7	5,684.7
usps	7,291	256	10	2.2
w7a	24,692	300	2	32.4
w8a	49,749	300	2	32.6

Investigación Científica 2016, the UAM–ADIC Chair for Data Science and Machine Learning and Instituto de Ingeniería del Conocimiento. We gratefully acknowledge the use of the facilities of Centro de Computación Científica (CCC) at UAM.

REFERENCES

- [1] H. He and E. A. Garcia, “Learning from imbalanced data,” *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, 2009.
- [2] B. Krawczyk, “Learning from imbalanced data: open challenges and future directions,” *Progress in AI*, vol. 5, no. 4, pp. 221–232, 2016.
- [3] B. Tang and H. He, “Gir-based ensemble sampling approaches for imbalanced learning,” *Pattern Recognition*, vol. 71, pp. 306–319, 2017.

Table V
TEST AUC FOR THE 0- AND 2- TO 5-HIDDEN LAYER MLPs CONSIDERED.

	MLPClassifier0	MLPClassifier2	MLPClassifier3	MLPClassifier4	MLPClassifier5
a4a	89.78 (5)	89.88 (3)	89.85 (4)	89.91 (2)	89.93 (1)
a8a	90.73 (5)	90.85 (1)	90.83 (2)	90.81 (3)	90.80 (4)
combined	92.31 (5)	94.82 (4)	94.84 (3)	94.92 (1)	94.89 (2)
dna	99.27 (1.5)	99.26 (3)	99.27 (1.5)	99.25 (4.5)	99.25 (4.5)
ijcnn1	91.34 (5)	99.33 (4)	99.34 (3)	99.53 (2)	99.54 (1)
letter	98.04 (5)	99.95 (3.5)	99.96 (1.5)	99.96 (1.5)	99.95 (3.5)
pendigits	99.08 (5)	99.87 (4)	99.88 (3)	99.90 (1)	99.89 (2)
satimage	96.28 (5)	98.87 (3.5)	98.90 (2)	98.87 (3.5)	98.92 (1)
shuttle	92.83 (5)	99.93 (4)	99.96 (3)	99.98 (2)	99.99 (1)
usps	99.11 (5)	99.58 (3.5)	99.58 (3.5)	99.65 (1)	99.60 (2)
w7a	95.40 (5)	96.31 (4)	96.34 (3)	96.58 (1)	96.50 (2)
w8a	95.99 (5)	96.20 (4)	96.22 (3)	96.36 (2)	96.71 (1)
mean rank	4.7	3.5	2.7	2.0	2.1

Table VI
RESULTS OF THE WILCOXON TESTS FOR MULTI-CLASS PROBLEMS.

	p-value	Hypothesis
MLPClassifier0 vs MLPClassifier2	0.00	Rejected
MLPClassifier0 vs MLPClassifier3	0.00	Rejected
MLPClassifier0 vs MLPClassifier4	0.00	Rejected
MLPClassifier0 vs MLPClassifier5	0.00	Rejected
MLPClassifier2 vs MLPClassifier3	0.12	Not rejected
MLPClassifier2 vs MLPClassifier4	0.02	Rejected
MLPClassifier2 vs MLPClassifier5	0.02	Rejected
MLPClassifier3 vs MLPClassifier4	0.03	Rejected
MLPClassifier3 vs MLPClassifier5	0.03	Rejected
MLPClassifier4 vs MLPClassifier5	0.82	Not rejected

- [4] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.
- [5] X. Liu, J. Wu, and Z. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 39, no. 2, pp. 539–550, 2009.
- [6] M. Galar, A. Fernández, E. B. Tartas, H. B. Sola, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Trans. Systems, Man, and Cybernetics, Part C*, vol. 42, no. 4, pp. 463–484, 2012.
- [7] P. M. Domingos, "Metacost: A general method for making classifiers cost-sensitive," in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999*, 1999, pp. 155–164.
- [8] Z. Zhou and X. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 1, pp. 63–77, 2006.
- [9] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 233–240.
- [10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.
- [11] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861 – 874, 2006, rOC Analysis in Pattern Recognition.
- [12] D. J. Hand and R. J. Till, "A simple generalisation of the area under the roc curve for multiple class classification problems," *Machine Learning*, vol. 45, no. 2, pp. 171–186, Nov 2001.
- [13] C. Ferri, J. Hernandez-Orallo, and M. Salido, "Volume under the roc surface for multi-class problems," in *Proceedings of 14th European Conference on Machine Learning*, 03 2003, pp. 108–120.
- [14] F. Provost and P. Domingos, "Tree induction for probability-based ranking," *Machine Learning*, vol. 52, no. 3, pp. 199–215, Sep 2003.
- [15] G. Tsoumakas, I. Katakis, and I. P. Vlahavas, "Mining multi-label data," in *Data Mining and Knowledge Discovery Handbook, 2nd ed.*, 2010, pp. 667–685.
- [16] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2008.
- [17] Y. Yang, "An evaluation of statistical approaches to text categorization," *Inf. Retr.*, vol. 1, no. 1-2, pp. 69–90, 1999.
- [18] J. Huang and C. X. Ling, "Using auc and accuracy in evaluating learning algorithms," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, no. 3, pp. 299–310, Mar. 2005.
- [19] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
- [20] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, Apr. 2011.
- [21] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, May 2010, pp. 249–256.
- [22] F. Chollet, "Keras: Deep learning library for theano and tensorflow," 2015. Available: <https://github.com/fchollet/keras>
- [23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [24] M. Lázaro, M. H. Hayes, and A. R. Figueiras-Vidal, "Training neural network classifiers through bayes risk minimization applying unidimensional parzen windows," *Pattern Recognition*, vol. 77, pp. 204 – 215, 2018.
- [25] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, no. 2-3, pp. 255–287, 2011.
- [26] S. González, S. García, M. Lázaro, A. R. Figueiras-Vidal, and F. Herrera, "Class switching according to nearest enemy distance for learning from highly imbalanced data-sets," *Pattern Recognition*, vol. 70, no. Supplement C, pp. 12 – 24, 2017.
- [27] S. Wang and X. Yao, "Multiclass imbalance problems: Analysis and potential solutions," *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 42, no. 4, pp. 1119–1130, 2012.
- [28] A. Achille and S. Soatto, "On the emergence of invariance and disentangling in deep representations," *CoRR*, vol. abs/1706.01350, 2017.
- [29] R. Vidal, J. Bruna, R. Giryes, and S. Soatto, "Mathematics of Deep Learning," *ArXiv e-prints*, Dec. 2017.

Chapter 5

Deep Support Vector Machines and Companion Losses

5.1 Summary

As mentioned, Support Vector Machines (SVM) are a very popular family of models that can achieve great performance in many supervised problems, specially when combined with kernels that give them non-linear capabilities. However, the use of kernels implies a much higher computational complexity both in training and inference, which results in SVMs not being appropriate in big data setups. The benefits of the margin-based loss functions employed in SVMs can, however, be transferred to MLP-based models. This idea is explored in this set of publications, obtaining more solid MLP models with non-linear capabilities that have a much better scalability with respect to the size of the dataset. Finally, a multi-loss MLP architecture featuring multiple outputs, one per loss function, that allows for combinations of traditional, margin-based and other problem-tailored losses, trained with standard back-propagation, is presented. Again, the flexibility of DNN development frameworks like Tensorflow [58] and the convenience of high-level libraries on top of them like Keras [66], compatible with general ML libraries like Scikit-learn [20], greatly simplified the experimental work.

5.2 Publications

We detail next the publications of this chapter, give their abstracts, and briefly list their main contributions for the sake of convenience.

Main publication

- David Díaz-Vico, Jesús Prada, Adil Omari, and José R. Dorronsoro. Deep support vector neural networks. *Integrated Computer-Aided Engineering*, pages 389–402, Jan 2020

Abstract: Kernel based Support Vector Machines, SVM, one of the most popular machine learning models, usually achieve top performances in two-class classification and regression problems. However, their training cost is at least

quadratic on sample size, making them thus unsuitable for large sample problems. However, Deep Neural Networks (DNNs), with a cost linear on sample size, are able to solve big data problems relatively easily. In this work we propose to combine the advanced representations that DNNs can achieve in their last hidden layers with the hinge and ϵ insensitive losses that are used in two-class SVM classification and regression. We can thus have much better scalability while achieving performances comparable to those of SVMs. Moreover, we will also show that the resulting Deep SVM models are competitive with standard DNNs in two-class classification problems but have an edge in regression ones.

Contributions: This article was published as an extension of [31]. The main improvements over the related congress paper were:

- A deeper analysis of the proposed model of a margin loss applied to DNN architectures
- A more thorough experimental study, incorporating several additional computer image datasets
- The conclusion that the use of the ϵ -insensitive loss function can help deep models achieve better results, specially when the objective metric is MAE.

Also, the conclusions obtained in [31] were consolidated. That is, the DSVM model is a valid tool to induce a margin while performing a non-linear transformation over the data and still be usable with quite large sample sizes, while Kernel SVMs can only operate when the sample size is relatively small.

Other publications

- David Díaz-Vico, Jesús Prada, Adil Omari, and José R. Dorronsoro. Deep support vector classification and regression. In José Manuel Ferrández Vicente, José Ramón Álvarez-Sánchez, Félix de la Paz López, Javier Toledo Moreo, and Hojjat Adeli, editors, *From Bioinspired Systems and Biomedical Applications to Machine Learning*, pages 33–43, Cham, 2019. Springer International Publishing

Abstract: Support Vector Machines, SVM, are one of the most popular machine learning models for supervised problems and have proved to achieve great performance in a wide broad of predicting tasks. However, they can suffer from scalability issues when working with large sample sizes, a common situation in the big data era. On the other hand, Deep Neural Networks (DNNs) can handle large datasets with greater ease and in this paper we propose Deep SVM models that combine the highly non-linear feature processing of DNNs with SVM loss functions. As we will show, these models can achieve performances similar to those of standard SVM while having a greater sample scalability.

Contributions: This congress paper introduced the Deep SVM (DSVM) models both for regression and classification, as the result of the use of margin-based losses from traditional SVMs being applied to MLP-based models. The implementation of such losses was possible thanks to the availability of computation

frameworks like Tensorflow [58], mainly oriented to the development of Deep Learning models, that incorporate automatic differentiation and other very useful functionalities. The results obtained in a selection of both regression and classification problems were competitive with those of Kernel SVMs, while the computational cost of the training algorithms scaled linearly with the size of the training dataset, indicating that the models proposed could also handle big data problems.

- David Díaz-Vico, Ángela Fernández, and José R. Dorronsoro. Companion losses for deep neural networks. In *Hybrid Artificial Intelligent Systems*, pages 538–549, Cham, 07 2021. Springer International Publishing

Abstract: Modern Deep Neuronal Network backends allow a great flexibility to define network architectures. This allows for multiple outputs with their specific losses which can make them more suitable for particular goals. In this work we shall explore this possibility for classification networks which will combine the categorical cross-entropy loss, typical of softmax probabilistic outputs, the categorical hinge loss, which extends the hinge loss standard on SVMs, and a novel Fisher loss which seeks to concentrate class members near their centroids while keeping these apart.

Contributions: This congress paper proposed a multi-loss MLP architecture taking advantage of the great flexibility of modern Deep Neural Network libraries like [66, 58, 59]. The use of multiple losses in classification represents yet another way to further adapt the model to the data, obtaining better predictions as a result. This was the case, in particular, of the combination of the traditional cross-entropy loss with the Fisher loss proposed in [29], which produced the best results.

Deep support vector neural networks

David Díaz-Vico^{a,*}, Jesús Prada^b, Adil Omari^c and José Dorronsoro^b

^a*Instituto de Ingeniería del Conocimiento, Universidad Autónoma de Madrid, Madrid, Spain*

^b*Departamento de Ingeniería Informática, Universidad Autónoma de Madrid, Madrid, Spain*

^c*Departamento de Teoría de la Señal, Universidad Carlos III, Madrid, Spain*

Abstract. Kernel based Support Vector Machines, SVM, one of the most popular machine learning models, usually achieve top performances in two-class classification and regression problems. However, their training cost is at least quadratic on sample size, making them thus unsuitable for large sample problems. However, Deep Neural Networks (DNNs), with a cost linear on sample size, are able to solve big data problems relatively easily. In this work we propose to combine the advanced representations that DNNs can achieve in their last hidden layers with the hinge and ϵ insensitive losses that are used in two-class SVM classification and regression. We can thus have much better scalability while achieving performances comparable to those of SVMs. Moreover, we will also show that the resulting Deep SVM models are competitive with standard DNNs in two-class classification problems but have an edge in regression ones.

Keywords: Support vector machines, deep learning

1. Introduction

Kernel Support Vector Machines (SVM; [1]) were very popular in the 1990s and early 2000's because of the powerful models they provided. However they are somewhat out of fashion in the current big data era. The main reason for this is their cost, at least quadratic (and often worse) with respect to sample size, in a sharp contrast with the linear cost in sample size of Deep Neural Networks (DNNs) that currently dominate big data machine learning (ML).

This cost is hard to avoid in the usual kernel formulation of SVMs. In fact, while at the end, SVMs are linear models on extended representations $\Phi(x)$ of the original patterns x , when working with effective kernels such as the Gaussian one, the representation lies in a Reproducible Kernel Hilbert Space (RKHS) and can only be handled implicitly through the dot products $\Phi(x) \cdot \Phi(x') = k_{\Phi}(x, x'; \Theta)$ where k_{Φ} is a suitable explicit kernel and Θ denotes possible kernel parameters. This implies that the SVM dual problem must be

solved. But its dimension is N , the sample size, one must work with an $N \times N$ kernel matrix and have to apply the SMO algorithm with a cost that very often will be at least quadratic in N (see Section 2 below). All this means that standard SVM training and, even more so, hyperparametrization will simply be too costly in a big data setting.

Many proposals have appeared in the literature to apply SVMs on large datasets, usually focused on Support Vector classification (SVC) problems. Among them we can mention incremental learning [2], ensemble learning [3] or the cutting planes method [4]. Nevertheless, it can be said that, unless substantial hardware resources are committed, current kernel SVM training methods are not time competitive for datasets with more than about 100,000 patterns. Recently, GPU compliant implementations of SVM training have been proposed [5] but, while greatly accelerating SVM training, the memory limitations of large sample SVMs still remain.

Larger datasets can be handled by working with linear SVMs using the LIBLINEAR library [6] over either the primal or dual problems, often using coordinate descent [7] to solve the minimization problems. However, efficient linear SVM models can only be expected when the original features have very large dimensions so that

*Corresponding author: David Díaz-Vico, Instituto de Ingeniería del Conocimiento, Universidad Autónoma de Madrid, C. Tomás y Valiente 11, Madrid, 28049, Spain. E-mail: david.diaz@iic.uam.es.

feature projections are no longer needed. When sample dimension is just moderate, linear SVM models are usually much less powerful than their Gaussian counterparts. Pegasos [8] can be used on a kernel setting but, as with LIBLINEAR, it must work with homogeneous models $w \cdot x$ or $w \cdot \Phi(x)$ without a bias term; this restricts the models to be homogeneous and while sometimes classification performance is comparable with that of full fledged SVMs [9], in other cases falls behind. Given the above difficulties in large problems, the possibility of bringing SVM training into a neural network setting has already been considered in [10], where it is compared with standard softmax DNNs on classification problems, and by [11] in speech recognition.

However, and in a different vein from these works, we first observe that the non differentiability of the SVM primals is rather mild; in fact, Pegasos exploits subgradient descent essentially by skipping the single non differentiability at 0 of the hinge loss, or at $\pm \epsilon$ of the ϵ -insensitive one. This is precisely the same approach followed on DNNs when having to handle the non differentiability of the ReLU activations while computing gradients by backpropagation. As a matter of fact, the hinge loss is already available in Keras [12], and the ϵ -insensitive loss can be written using, say, TensorFlow primitives; in turn, this makes possible to perform backpropagation on that loss.

This opens the way to try to solve the primal problem directly working with an **explicit** kernel transformation $\Phi(x; \mathcal{W})$ parameterized by a certain weight set \mathcal{W} which, in turn, can be obtained through some training process. Of course, an appropriate Φ has to be found and a natural choice is the transformation performed by a DNN between its inputs and its last hidden layer; we will call the resulting models Deep SVMs and our proposal in this work is to build Deep SVM models in much the same way that DNNs are now routinely trained.

Notice that, at the end, this point of view also leads to a linear model acting on a hidden representation. However, and although seemingly close, the final models will be quite different. Following the terminology in [13], DNNs belong to the category of iterative summarization methods; on the other hand, Gaussian SVMs, the most popular way to derive non linear SVMs, are placed in [13] within the case-based reasoning methods, jointly for instance with k -NN methods. In fact, Gaussian SVMs can be seen as covering the sample space with generalized balls centered on the learned support vectors and, similarly to k -NN methods, they cannot handle very large samples. However,

and as pointed out in [13] and often observed in the literature (see for instance [14]), when properly hyperparameterized, different ML methods have similar performances in problems with features that have clear predictive meanings. This suggests to compare first the performance of Gaussian and Deep SVMs, which we did in our previous work [15] and which we summarize below for the reader's convenience.

In any case, Gaussian SVM comparisons are only possible for datasets up to a given size. In fact, as of today, they have substantial train and, more so, hyperparameterization costs with sample sizes above 10^5 and, thus, they are no match for Deep SVMs on problems of such sizes. But this is far from implying that Deep SVMs are then to be the preferred model in that setting for then another obvious adversary arises, namely standard DNNs with cross entropy loss for classification and squared error loss for regression. First, though, it must be observed that, contrary with what happened with Gaussian SVMs, Deep SVMs and standard DNNs work at first sight in a very similar fashion. In fact, two learning processes take place simultaneously. First, a non linear representation $z = \Phi(x, \mathcal{W}^*)$ between the original inputs and their last hidden layer (LHL) representation is learned. On the other hand, the network also learns a linear model $y = w^* \cdot z + w_0^*$ between these LHL representations and the network outputs y .

The difference lies, however, in their respective training goals. On standard DNNs the goal is the estimation of posterior class distributions in classification or to mean squared error minimization in regression. On the other hand, hinge loss minimization in our proposed Deep SVM classifier training means that a maximum margin model is learned on the last hidden layer; similarly, the ϵ insensitive loss achieved in the training of our proposed Deep SVM regressors implies that an optimal ϵ tube is fitted around the model built in the last hidden layer. Those are quite different training objectives with possible advantages of their own, as large margins should result in good generalization and the ϵ parameter gives extra flexibility to the deep support vector regressors.

We will focus here our attention in the two-class classification and regression performance of fully non linear DNNs versus that of Deep SVMs. We will first work with feed-forward fully connected architectures for classification and regression but we will also consider image classification datasets where we will use problem specific architectures already proposed in the literature and whose results are close to the state of the art. In summary, our main contributions are:

- The proposal of Deep SVMs with the same ar-

chitectures and training procedures of either standard or advanced DNNs but using the hinge and ϵ -insensitive losses typical of SVMs for two-class classification and regression respectively.

- A statistical comparison of the performance of DNNs and deep SVMs with standard, fully connected feedforward architectures in mid size classification and regression problems.
- A statistical comparison of the performance of DNNs and Deep SVMs over large image datasets, using problem specific advanced deep architectures that are representative of the current state of the art.

In other to streamline the presentation, we will use the acronyms **DNNC** and **DNNR** (Deep Neural Network Classification and/or Regression) to refer to models with a DNN architecture and the cross entropy or mean squared error losses in classification and regression respectively. Similarly, we will use the acronyms **DSVC** and **DSVR** (Deep Support Vector Classification and/or Regression) to refer to models with a DNN architecture but using now the hinge loss for two-class classification and the ϵ -insensitive loss for regression. Finally we will refer to Gaussian SVM models for two class classification as **GSVC** and to Gaussian SVM models for regression as **GSVR**.

Putting together our results here and those in [15], a first conclusion is that GSVCs, DNNCs and our proposed DSVCs have similar performances in two-class classification problems. However, the situation for regression problems is different as we will numerically show that DSVRs have an hedge over DNNRs, not only when mean absolute error (MAE) is the cross validation (CV) score used for model hyperparametrization but also when using mean squared error (MSE) as the CV score.

The rest of the paper is organized as follows. We give a short unified exposition to standard SVMs in Subsection 2, where we also address training and test complexity and model sparsity. We present our deep SVM proposal in Section 3, where we also briefly discuss training complexity. For the sake of the reader, we briefly recover our results in [15] and Sections 5 and 6 contain our comparison of standard DNNs versus Deep SVMs. More precisely, in Section 5 we will compare, using the same datasets of Section 4, standard fully connected DNNCs and DSVCs first and then DNNRs and DSVRs. In both cases we will also address the statistical significance of our results. On the other hand, in Section 6 we shall work with three large image datasets, the MNIST, MNIST Fashion and CIFAR 10 problems.

For the first two we will use the well known LeNet-5 architecture [16] to compare the cross entropy and hinge losses (i.e., the corresponding DNNC and DSVC models) on the ten 2-class problems of distinguishing on MNIST each of the 0 to 9 digits from the others, and similarly for MNIST Fashion. We will follow the same approach with the ten class CIFAR 10 problems but here the cross entropy and hinge losses will act on top of a ResNet32 V1 model [17], a substantially more complex deep residual network. The comparisons will also include LIBLINEAR models and we will address their statistical significance. Finally, the paper ends with a discussion, conclusions and pointers to further work.

2. A SVM review

2.1. Primal and dual SVC and SVR

We will briefly review here primal and dual SVC and SVR in a unified view following [18]. Assume a sample given as a set of triplets $\mathcal{S} = \{(x_i, y_i, p_i) : i = 1, \dots, N\}$ with $y_i = \pm 1$ and p_i some scalar values, and consider the minimization problem:

$$\begin{aligned} \min_{w, b, \xi} \mathcal{P}(w, b, \xi) &= \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \\ \text{s.t.} \quad &\begin{cases} y_i(w \cdot x_i + b) \geq p_i - \xi_i, & \forall i, \\ \xi_i \geq 0 & \forall i. \end{cases} \end{aligned} \quad (1)$$

Now if we take $p_i = 1$ for $i = 1, \dots, N$, Eq. (1) reduces to the standard two-class Support Vector Classification (SVC) problem

$$\begin{aligned} \min_{w, b, \xi} \mathcal{P}_C(w, b, \xi) &= \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad &\begin{cases} y_i(w \cdot x_i + b) \geq 1 - \xi_i, & \forall i, \\ \xi_i \geq 0, & \forall i. \end{cases} \end{aligned} \quad (2)$$

On the other hand, given an initial sample $\mathcal{R} = \{(x_i, t_i) : i = 1, \dots, N\}$ and enlarging it as $\mathcal{S} = \{(x_i, y_i, p_i) : i = 1, \dots, 2N\}$ by taking, for $1 \leq i \leq N$, $y_i = 1$, $p_i = t_i - \epsilon$ and $y_{N+i} = -1$, $p_{N+i} = -t_i - \epsilon$, and $x_{N+i} = x_i$, Eq. (1) reduces now to

$$\min_{w, b, \xi, \xi'} \mathcal{P}_R(w, b, \xi) = \frac{1}{2} \|w\|^2 \quad (3)$$

$$+ C \sum_{i=1}^N (\xi_i + \xi'_i) \quad (4)$$

$$\text{s.t.} \quad \begin{cases} w \cdot x_i + b - t_i \leq \epsilon + \xi_i, & \forall i, \\ t_i - w \cdot x_i - b \leq \epsilon + \xi'_i, & \forall i, \\ \xi_i \geq 0 & \forall i, \\ \xi'_i \geq 0 & \forall i, \end{cases}$$

which is just the standard formulation of the ϵ -insensitive SV regression (SVR).

While its loss is not differentiable, Eq. (1) is, nevertheless, a convex minimization problem. Because of this, one does not attempt to solve it directly; instead, standard Lagrangian optimization is applied to reduce Eq. (1) to the equivalent problem of minimizing the general dual function

$$\begin{aligned} \min_{\alpha} \mathcal{D}(\alpha) &= \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j Q_{ij} \\ &- \sum_i \alpha_i p_i \\ \text{s.t.} \quad &\begin{cases} 0 \leq \alpha_i \leq C \quad \forall i, \\ \sum_i \alpha_i y_i = 0, \end{cases} \end{aligned} \quad (5)$$

where Q is the matrix with entries $Q_{ij} = y_i y_j x_i \cdot x_j$.

In any case, a clear drawback of trying to solve Eq. (1) directly is that it would result in plain linear models, possibly not powerful enough to deal with the problems at hand. The dual formulation overcomes this as it can be stated in a kernel setting by replacing the inner products $x_i \cdot x_j$ in Q with a kernel function $k(x_i, x_j)$. More precisely, for a general kernel, such as the usual Gaussian one, it can be shown that there is a map Φ projecting the initial patterns into an abstract reproducible kernel Hilbert space (RKHS) [19] such that we have $k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$. The final models will be still linear but no longer in the original features but on their kernel defined extensions $\Phi(x)$. However, the exact knowledge of Φ is not needed and can be implicitly handled.

2.2. Complexity and sparsity of kernel SVMs

Notice that the dual problem has to be solved in a space with dimension N , i.e., sample size. It involves having to handle the kernel matrix, with size $N \times N$, which is bound to be problematic for large samples. Moreover, the standard algorithm to solve Eq. (5) is Standard Sequential Minimization, SMO, which proceeds iteratively, with each iteration having an $O(N)$ cost. In turn, the number of iterations is $\Omega(N_{SV})$, with N_{SV} the number of support vectors, which in most problems is $\Theta(N)$. Thus, SMO training will require at least $\Theta(N)$ iterations and has, hence, a minimum cost $\Omega(N^2)$ which is quite higher in practice, sometimes being estimated to be cubic in N . It is thus clear that for large samples (such as those with more than 10^5 samples), SVM training may prove too costly.

Moreover, an often cited property of SVMs is that their solution is sparse, i.e., the number of nonzero

optimal α_i^* values (and of the corresponding support vectors) is less than N ; however, it is also often the case that the number of support vectors (i.e., patterns x_i for which $\alpha_i^* > 0$) is still $\Theta(N)$ which hinders, for instance, the use of SVMs in near on line settings, as its application of a kernel SVM model on new patterns has then a $\Theta(N)$ cost, as all support vectors enter the model.

3. Deep SVC and SVR

It is well known that large margins usually imply better generalization. For linearly separable problems and a C large enough, it is also well known that linear SVC models achieve a maximum margin. But this is not necessarily so when the cross entropy loss of Logistic Regression is used; an example can be seen in Fig. 1. It shows the SVC (red) and Logistic Regression (blue) separating hyperplanes built over linearly separable samples given by two Gaussians centered at $(0, 1)$ (top) and $(0, -1)$ (bottom); class sizes are 100 for the top class and 500 for the bottom one. Gaussian standard deviation is 0.25 in the left figure and 0.33 in the right one. As it can be seen, the SVC hyperplane (red) achieves a maximum margin while the cross entropy hyperplane (blue) is much closer to the minority top class. (The points of each class closest to the SVC hyperplane are shown as red dots.)

Of course, the usefulness of linear classifiers is rather small and, as we have just discussed, the standard way to introduce non linearity into SVMs is to use the kernel trick. However, the price to pay for this is the need to solve the dual problem, with a cost usually at least quadratic with respect to sample size. But, in any case, we point out that the loss in Eq. (1) can be written as

$$\max\{0, p - y(w \cdot x + b)\}.$$

For classification this is just the hinge loss

$$h(y, f(x)) = \max\{0, 1 - yf(x)\}$$

with $f(x) = w \cdot x + b$, while for regression we have the ϵ -insensitive loss

$$\ell_{\epsilon}(y, f(x)) = \max\{0, |y - f(x)| - \epsilon\}.$$

Observe that in both cases the non differentiability of the primal loss is rather mild and in fact very similar to that of the ReLU activations routinely handled in Deep Neural Network (DNN) packages. Moreover, if \mathcal{W}_h is the set of weights between the input and the last hidden layer in a DNN, the input to last hidden layer

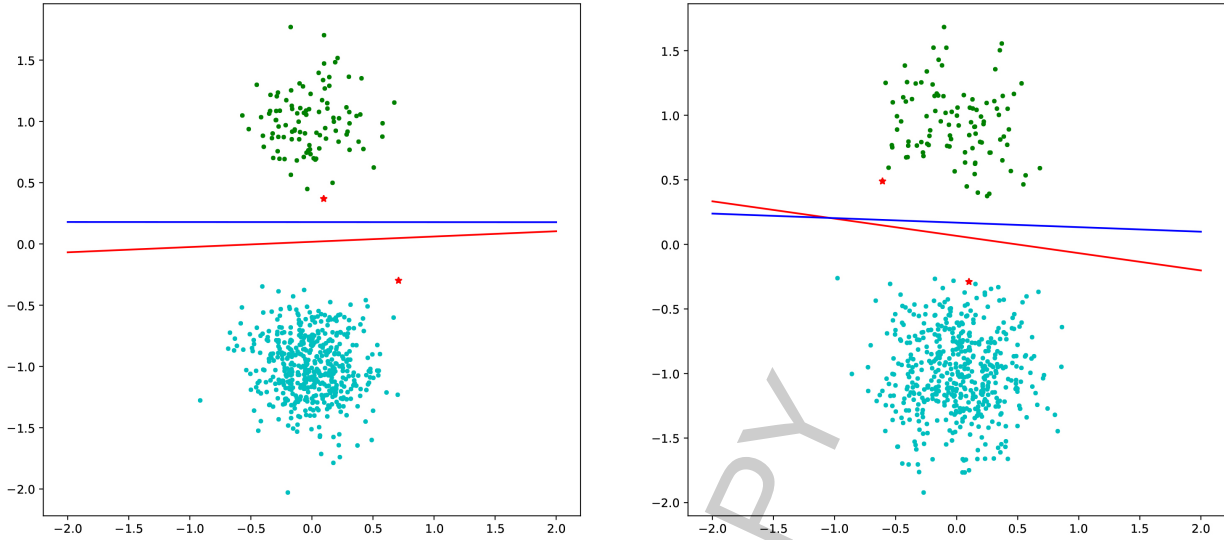


Fig. 1. SVC (red) and Logistic Regression (blue) classifiers for two class linearly separable problems. The SVC hyperplane has a maximum margin while the Logistic Regression one is closer to the minority class.

transformation $\Phi(x) = F(x, \mathcal{W}_h)$ that such a DNN performs usually results in much enhanced pattern representations over which linear models may yield powerful classifiers and regressors. In particular, using a flexible enough DNN would results in a linearly separable problem in the last layer and, as illustrated in Fig. 1, a maximum margin classifier when the hinge loss is used, something that would not happen with the cross entropy loss usually applied in two class DNN classification. All this suggests to exploit this explicit kernel to avoid dealing with the SVM dual problem.

More precisely, consider a DNN with linear outputs, a weight set \mathcal{W}_h between the input and the last hidden layer, and linear weights and bias (w, b) from the last layer to the network outputs. We denote the input-output transformation of such a network as

$$\begin{aligned} f(x, w, b, \mathcal{W}_h) &= w \cdot F(x, \mathcal{W}_h) + b \\ &= w \cdot \Phi(x) + b \end{aligned}$$

where $F(x, \mathcal{W}_h) = \Phi(x)$ is the last hidden layer representation of x . The network's cost function will be

$$\begin{aligned} J(w, b, \mathcal{W}_h) &= \frac{1}{N} \sum_{i=1}^N \ell(y_i, w \cdot F(x_i, \mathcal{W}_h) + b) \\ &\quad + \alpha_S \|w\|^2 + \alpha_H \mathcal{R}(\mathcal{W}_h), \end{aligned} \tag{6}$$

where $\ell(y, \hat{y})$ is the loss that we want to use, and we split weight regularization in two parts, the squared norm $\|w\|^2$ of the linear output weight vectors and the

sum $\mathcal{R}(\mathcal{W}_h)$ of the Frobenius norms for the weight matrices between hidden layers. We shall consider the hinge loss for two-class classification problems and the ϵ -insensitive loss for regression ones. Now, the weight gradients of such a network can be computed by standard backpropagation and then the network trained by plain gradient descent, as in Algorithm 1, or, as we will do later on, using the Adam solver [20].

Algorithm 1: Deep SVM backpropagation

Input: Random initial weights $W = (W_\ell), 1 \leq \ell \leq L$, dataset (X, Y) , learning rate sequence ρ_t

- 1 For epoch t in $1 \dots T$:
- 2 While (X, Y) is not empty:
- 3 Select a batch (X', Y') from (X, Y) without replacement
- 4 # Forward Pass
- 5 $Z_0 = X'$
- 6 For layer ℓ in $1, \dots, L$:
- 7 $Z_\ell = f_\ell(Z_{\ell-1} W_\ell)$ # f_ℓ : activation at layer ℓ
- 8 # Backward Pass
- 9 $E_L = \mathcal{L}(Y', Z_L)$ # \mathcal{L} either hinge or ϵ -insensitive loss
- 10 For layer ℓ in $L-1, \dots, 1$:
- 11 $S_\ell = Z_{\ell-1} W_\ell$
- 12 $E_\ell = \frac{\partial E_{\ell+1}}{\partial S_\ell}$
- 13 $\nabla_{W_\ell} \mathcal{L} = E_\ell \odot Z_{\ell-1}$
- 14 $W_\ell = W_\ell - \rho_t \nabla_{W_\ell} \mathcal{L}$

Output: Fitted weights $W = (W_\ell)$

Assume that such a network has been trained, let $(w^*, b^*, \mathcal{W}^*)$ be the optimal weights and denote as $z^* = F(x, \mathcal{W}^*)$ the network's last hidden layer outputs. Then (w^*, b^*) solve the problem of minimizing

$$\frac{1}{N} \sum_{i=1}^N \ell(y_i, w \cdot (z^*)_i + b) + \alpha_S \|w\|^2 \quad (7)$$

over w, b : if not and w', b' were a better weight-bias pair, the overall weight set (w', b', \mathcal{W}^*) would improve on $(w^*, b^*, \mathcal{W}^*)$. We can rewrite Eq. (7) as

$$\begin{aligned} & \operatorname{argmin}_{w,b} \frac{1}{2} \|w\|^2 \\ & + \frac{1}{2\alpha_S N} \sum_{i=1}^N \ell(y_i, w \cdot (z^*)_i), \end{aligned} \quad (8)$$

which is just the problem solved in SVC when ℓ is the hinge loss and the problem solved in SVR when ℓ is the ϵ -insensitive loss. Thus, after a DNN has been trained to solve Eq. (7), the optimal output weights w^* and bias b^* solve either the SVC or SVR problems over the network outputs at the last hidden layer. In summary, we can couple DNN architectures with linear outputs and either the hinge or ϵ -insensitive losses to bring SVC or SVR into a DNN set up; we will call DSVC or DSVR the resulting deep SVM models.

The cost of training them is just that of training a standard DNN, which depends on three inputs. The first is the architecture or, more precisely, the number n_W of network weights, the second is the number n_I of iterations to be performed by the solver and the third is sample size N . Obviously, n_W is independent of N and, contrary to what happens in SMO, so is n_I , which basically depends on the precision to which the DNN cost function is to be minimized. Putting all this together, DSVC or DSVR training has a

$$O(n_W \times n_I \times N)$$

cost, which now grows linearly with N but not quadratically, as is the case with SVMs. Moreover, the cost of applying the model on a single pattern is now $O(n_W)$, independent of sample size and, hence, much faster in principle than that of a SVM model.

In any case, the classification and regression performance of such networks, which we will call Deep SVMs, has to be compared with that of standard Gaussian SVC and SVR. This was done in [15] and we will briefly recall it in Section 4. However, and as mentioned before, even if training Gaussian SVMs on large datasets is, as of today, too costly and deep SVM models can be an alternative, it is clear that in the large sample setting there is an obvious counterpart to Deep SVMs, precisely standard cross entropy or mean squared error DNNs. We will compare them with Deep SVMs in Sections 5 and 6.

Table 1

Number of train and (when available) test patterns and dimensions in the two-class (top) and regression (bottom) problems

	n. patterns train	n. patterns test	dimension
a4a	4781	27780	123
a8a	22696	9865	123
australian	690		14
cod-rna	59535	271617	8
diabetes	768		8
german.numer	1000		24
ijcnn1	49990	91701	22
w7a	24692	25057	300
w8a	49749	14951	300
abalone	4177		8
bodyfat	252		14
cadata	20640		8
cpusmall	8192		12
housing	506		13
mg	1385		6
mpg	392		7
space_ga	3107		6

4. Gaussian versus deep SVMs

In this section we will briefly summarize for the convenience of the reader the comparisons between Gaussian two class classification (GSVC) and regression (GSVR) models and their deep counterparts DSVC and DSVR given in [15], where we used for comparison the classification datasets a4a, a8a, australian, cod-rna, diabetes, german.numer, ijcn1, w7a and w8a, and the regression datasets abalone, bodyfat, cadata, cpusmall, housing, mg, mpg and space_ga. All are taken from the LibSVM repository [21] (which includes references to papers where they have been used) and their training and, when available, test sample sizes and dimensions are given in Table 1.

We compared GSVC and GSVR models against DSVC and DSVR networks with 1, 3 and 5 hidden layers to be trained using the adam optimizer over 200 pattern minibatches; more details are given in [15]. For GSVCs we must choose optimal values of two hyperparameters, the regularization constant C and the Gaussian kernel width γ ; we must add an extra hyperparameter ϵ for GSVR. In both cases we considered 5 logarithmically equi-spaced C values in the interval $[10^{-3}, 10^6]$ and γ values of the form $\frac{2^k}{d}$, with d pattern dimension and $-3 \leq k \leq 6$. Before this we scaled inputs feature-wise to a $[0, 1]$ range. For ϵ , we considered for GSVR 5 equi-distributed log values in the interval $[2^{-10} \times \text{std}(y), 2^{-1} \times \text{std}(y)]$. For DSVC and DSVR models we chose optimal values of the hyperparameters α_S and α_H by exploring now 5 evenly log spaced values in the interval $[2^{-30}, 2^{10}]$; the ϵ DSVR parameter

Table 2
Accuracies in the two-class problems (taken from [15])

	SVC	DSVC1	DSVC3	DSVC5	DSVC
a4a	84.32	84.19	84.19	84.27	84.27
a8a	84.92	85.18	85.14	85.14	85.18
australian	85.50	85.51	86.82	85.21	86.82
cod-rna	96.58	96.66	96.61	96.45	96.66
diabetes	77.60	76.95	76.43	77.21	77.47
german.numer	76.10	75.80	75.10	75.80	75.80
ijcnn1	97.93	98.88	99.07	98.99	99.07
w7a	98.87	98.82	98.83	98.83	98.83
w8a	99.04	98.99	99.16	99.03	99.16

was chosen again within 5 equi-distributed log values in the interval $[2^{-10} \times \text{std}(y), 2^{-1} \times \text{std}(y)]$.

In the classification problems having a separate test set available, GSVC and DSVC model hyperparameters were found by 4-fold stratified cross validation (CV) over the train set using accuracy as the CV score; we then computed test accuracies over the test set. When no test set is available (i.e., in three classification and all regression datasets), model performance was measured by a nested, two loop, 4-fold CV approach, in which each one of the four outer folds is kept for testing and the other three folds are passed to the inner loop, where best hyperparameters are obtained again by 4-fold CV and tested on the outer test folds. The regression score was the mean absolute error (MAE).

We report in Table 2 the GSVC accuracies and those of DSVC models with with 1 (DSVC1), 3 (DSVC3) and 5 (DSVC5) layers. Similarly, we report now in Table 3 the GSVR mean absolute errors (MAEs) and those of DSVR models with 1 (DSVR1), 3 (DSVR3) and 5 (DSVR5) layers. For a better reading, in both tables we give in the last column the accuracy or MAE of the best performing DSVC or DSVR model. As it can be seen, GSVCs give the largest accuracies on the four datasets, a4a, cod-rna, german.numer and w7a while for the other five datasets the highest accuracy is achieved by a DSVC model. In regression, GSVRs give the smallest MAE on the four datasets, bodyfat, cpusmall, housing and mpg and a DSVR model gave the smallest MAE for the other four datasets. As concluded in [15], the performance of GSVC and GSVR models, on the one hand, and DSVC and DSVR models is fairly balanced.

To finish this Section we will briefly discuss time comparisons between Gaussian and Deep SVM models. We have performed execution time experiments for both, but we observe first that measuring times is not as straightforward as it may look, for we are using two quite different implementations. For standard SVMs we use the scikit-learn implementation that has at its

Table 3
MAEs in the regression problems (taken from [15])

	SVR	DSVR1	DSVR3	DSVR5	DSVR
abalone	1.48	1.49	1.50	1.51	1.49
bodyfat ($\times 100$)	0.05	0.42	0.51	0.28	0.28
cadata ($/ 10000$)	4.96	4.81	3.26	3.26	3.26
cpusmall	2.13	2.21	2.19	2.40	2.19
housing	2.28	2.58	2.34	2.57	2.34
mg ($\times 100$)	9.26	9.68	9.13	9.58	9.13
mpg	1.91	2.39	2.49	2.36	2.36
space_ga ($\times 100$)	9.67	9.14	8.63	8.86	8.63

Table 4

Train and test times in minutes for the larger regression (top) and classification (bottom) datasets

	tr_svm	tr_dnn	tr_mlp	ts_svm	ts_dnn	ts_mlp
abalone	0.0016	0.2117	0.0380	0.0006	0.0016	0.0001
cadata	0.2840	0.5840	0.2481	0.0206	0.0051	0.0007
cpusmall	0.0462	0.3995	0.0599	0.0051	0.0023	0.0003
space_ga	0.0094	0.1652	0.0126	0.0005	0.0014	0.0001
a4a	0.0241	0.3059	0.1053	0.1035	0.0142	0.0024
a8a	0.5986	0.4780	0.4452	0.1819	0.0055	0.0007
w7a	0.2687	0.7161	0.1755	0.1759	0.0134	0.0020
w8a	11.9484	2.4366	0.3211	0.7897	0.0081	0.0013
ijcnn1	0.2278	2.4964	0.5845	0.2615	0.0387	0.0076
cod-rna	0.4152	3.2763	0.7782	0.6527	0.1082	0.0194

core the extremely efficient C++ implementation of the LIBSVM library. On the other hand, for deep SVMs we rely on the Keras framework with the Adam solver [20], which, in turn, relies on TensorFlow library. While quite efficient, notice that TensorFlow is a general purpose deep network library and, thus, not specialized in any particular kind of deep architecture. In fact, the loss gradients are derived through automatic differentiation procedures which work with very general architectures and losses. However, while this is very convenient, it must also necessarily hamper execution times; to all this, Keras adds its own overhead. To have an intermediate yardstick we have also measured train and test times of our fully connected models when using the MLPClassifier and MLPRegressor classes in the scikit-learn library. They are implemented in Python and Cython and while being thus less efficient than the compiled LIBSVM library, they are more compactly programmed than the Keras models (although much less general), which results in shorter training times.

With all those caveats, we give in Table 4 for the largest regression (top) and classification (bottom) datasets, timings for train (tr_svm/dnn/mlp columns) and test (ts_svm/dnn/mlp columns), which correspond to LIBSVM, Keras and scikit-learn MLPs, respectively. Model hyperparameters for each dataset are those corresponding to the best cross validation estimators just described. As it can be seen, SVM training times are

Table 5
Accuracy comparisons of one, three and five hidden layer models when hyperparameterized with hinge (DSVC) and binary cross-entropy (DNNC) scores. Larger accuracies in bold face

	1 hid. layer		3 hid. layers		5 hid. layers	
	DSVC	DNNC	DSVC	DNNC	DSVC	DNNC
a4a	84.46	84.59	83.95	84.48	84.39	84.42
a8a	85.08	85.38	85.43	84.96	84.81	80.38
australian	85.64	87.96	85.36	85.94	85.50	86.66
cod-rna	96.56	96.30	96.56	96.70	96.23	94.59
diabetes	77.34	76.30	77.47	76.82	77.73	76.17
german.numer	75.10	74.80	73.10	75.90	74.50	75.80
ijcnn1	98.59	98.61	98.53	99.03	98.70	96.07
w7a	98.90	98.83	98.89	98.68	98.73	98.84
w8a	99.35	99.30	99.24	99.38	99.20	99.33

of the order of magnitude of those of MLPClassifier or MLPRegressor. The exceptions are a4a (a rather small dataset) and w8a (where SVMs take too long). As expected, Keras times are longest but they catch up as sample size increases. But notice also that for all datasets but the relatively small abalone and space_ga, LIBSVM test times are about 10 times longer than Keras' test times, and near or above 50 times those of MLPClassifier or MLPRegressor. Therefore, and technology and implementation issues aside, these performances essentially confirm what is to be expected on theoretical grounds, as discussed at the end of Sections 2 and 3.

5. Standard DNNs versus deep SVMs

Recall that for large datasets the counterpart to our DSVC and DSVR proposals are standard deep networks for either two-class classification with binary cross entropy as the training loss and sigmoid outputs (i.e., DNNC in our notation) or for regression with the squared loss for training and linear outputs (i.e., DNNR in our notation). In both cases we will use Keras implementations of networks with one, three and five hidden layers, and linear outputs. As in [15], these have 100 units on the intermediate ones and $0.1 \times |S|$ units in the last hidden layer, with $|S|$ denoting sample size and a lower bound in layer size of 100 and an upper bound of 1000. In other words, the size of the last hidden layer is $\min\{1000, \max\{100, 0.1 \times |S|\}\}$. Similarly, the DSVC and DSVR losses will be hinge and ϵ -insensitive, respectively.

Also, and as before, Tikhonov regularization will be applied in all layers, with two different parameters, α_S for the output weights and α_R in all others; for DSVR models we will also hyperparameterize the ϵ loss param-

eter. We will change slightly the α_S , α_R and ϵ ranges of the previous section. The α_S and α_R values will be of the form 10^k , $-5 \leq k \leq 2$, while the ϵ values will be $2^{-k} \times \sqrt{0.5}$, $-4 \leq k \leq 0$. Inputs will be scaled feature-wise to zero mean and one standard deviation. For regression problems we will also use a scikit-learn pipeline that incorporates a TransformedTargetRegressor component so that targets are normalized to zero mean and one standard deviation before training and de-normalized afterwards.

We will work with the same CV based hyperparameterization scheme of the previous section but for classification we will use here the $f1$ score for hyperparameter selection, as it involves both recall and precision and gives a more balanced performance measure. We report accuracy values in Table 5, where for easier reading we put in boldface the largest value for each architecture. This has only a descriptive value and a more relevant comparison could be made applying statistical significance results. To try to obtain such a slightly more precise comparison this way, we will pair for each problem and DNN architecture the DSVC and DSVR test accuracies. Since there are 9 dataset and 3 DNN architectures, this gives in principle a paired sample with 27 values; after this grouping, the accuracy averages are 88.75 ± 9.02 for DNNCs and 88.86 ± 9.13 for DSVCs, very similar. We can apply a Wilcoxon signed rank test on these 27 test accuracies, but with the clear caveat that these accuracies are probably not independent, as the three models for each problem share the same test set. Thus the following results must be seen in this light. It turns out that the resulting value is $p = 0.99$; if the test accuracies were independent, this would imply that we cannot reject the null hypothesis of the two paired samples come from the same distribution. Therefore, we could conclude that the performance of DNNC and DSVC models is very similar. However, as

Table 6

Mean absolute errors and p values for DSVR and DNNR models with one, three and five hidden layers. Smaller values in bold face when absolute error distributions are statistically different

	1 hid. layer			3 hid. layers			5 hid. layers		
	DSVR	DNNR	p	DSVR	DNNR	p	DSVR	DNNR	p
abalone	1.46	1.47	0.04	1.46	1.45	0.82	1.46	1.47	0.00
bodyfat ($\times 10^2$)	0.13	0.16	0.00	0.17	0.14	0.09	0.13	0.13	0.58
cadata ($/ 10^4$)	3.54	3.76	0.00	3.33	3.48	0.00	3.29	3.55	0.00
cpusmall	2.13	2.16	0.24	2.06	2.08	0.36	2.04	2.20	0.00
housing	2.25	2.66	0.00	2.04	2.18	0.02	2.07	2.10	0.44
mg ($\times 10^2$)	9.08	9.25	0.09	9.60	9.18	0.00	9.23	9.38	0.16
mpg	1.90	1.95	0.04	2.01	2.04	0.66	2.04	2.04	0.93
space_ga ($\times 10^2$)	0.08	0.08	0.64	0.07	0.07	0.31	0.07	0.07	0.32

Table 7

Mean squared errors and p values for DSVR and DNNR models with one, three and five hidden layers. Smaller values in bold face when squared error distributions are statistically different

	1 hid. layer			3 hid. layers			5 hid. layers		
	DSVR	DNNR	p	DSVR	DNNR	p	DSVR	DNNR	p
abalone	4.51	4.39	0.35	4.53	4.35	0.88	4.57	4.46	0.00
bodyfat ($\times 10^4$)	0.11	0.13	0.00	0.13	0.13	0.14	0.13	0.14	0.54
cadata ($/ 10^8$)	28.88	30.14	0.00	27.49	27.24	0.00	26.37	26.95	0.00
cpusmall	8.79	8.86	0.32	8.12	8.18	0.52	8.24	8.62	0.00
housing	13.35	17.05	0.00	10.50	11.41	0.03	10.14	10.27	0.40
mg ($\times 10^4$)	147.58	144.26	0.15	164.54	144.89	0.00	158.91	146.88	0.90
mpg	7.19	7.41	0.01	7.96	8.08	0.64	8.35	8.11	0.91
space_ga ($\times 10^4$)	107.82	105.24	0.82	103.49	101.61	0.53	101.57	93.32	0.62

said before, accuracy independence is not guaranteed, and the preceding can only be taken as a hint of similar performance.

The situation in regression will be slightly different. Notice that if we retain MAE as the validation score, as done in [15], this would help the DSVR models in detriment of DNNR models that aim to minimize the squared error. And inversely, if mean squared error (MSE) is used as the validation score, DNNRs would have then the advantage over DSVRs. Because of this we will compare DNNRs and DSVRs hyperparametrized using both MAE and MSE for CV scoring. Of course, this will result in possibly different hyperparameters but also on a fairer model comparison.

MAE scores are given in Table 6 and MSE ones in Table 7. Both tables also give p values which have been derived as follows. While, in classification, network outputs are only indirectly related with, say, accuracies, in regression the MAE and MSE values are just the averages of the absolute and square error per pattern. This suggest to perform model comparison by testing the distributions of the absolute and squared errors for each dataset and model. The p values of both tables correspond to the application of Wilcoxon signed ranked test on the error distributions of model outputs over the test sub-folds; recall that we apply here the previously described nested 4-fold CV scheme.

Both tables show in bold face the smallest MAE or MSE values when the null hypothesis of the error distributions being the same can be rejected at the $p = 0.05$ level. As it can be seen in Table 6, DSVRs with 1 hidden layer give a smaller MAE in 5 datasets, in 2 datasets when they have 3 hidden layers and in 3 datasets when having 5 layers; on the other hand, DNNRs give a smaller MAE in just one problem using three hidden layers. We also put in italics the smallest values over the three architectures considered when it is significant at the 0.05 level; this is achieved in 5 problems by a DSVR models and in none by a DNNR one. Thus, DSVR have a small advantage here.

The situation is more balanced with respect to MSE values. As it can be seen in Table 7, DSVRs with 1 hidden layer give a smaller MSE in 4 datasets, in 1 dataset when they have 3 hidden layers and in 2 datasets when having 5 layers; in contrast, DNNRs gives a smaller MSE in two problems using three hidden layers and in one using five. We put also here in italics the smallest values over the three architectures considered when it is significant at the 0.05 level; this is achieved by a DSVR model in 3 problems and in one by a DNNR one. While DSVR also seem to the advantage here, is clearly smaller than in the case of MAEs.

Summing things up, we can say that the classification performance of DNNC and DSVC models is fairly

Table 8

Number of train and test patterns and dimension in one versus rest MNIST, MNIST fashion and CIFAR 10 problems

	n. patterns train	n. patterns test	Dimension
MNIST	60,000	10,000	1,024
MNIST fashion	60,000	10,000	1,024
CIFAR 10	50,000	10,000	3,072

similar but, on the other hand, DSVR models seem to perform clearly better for regression, when MAE is the preferred metric and also slightly so when MSE is considered. As a possible reason for this is that DSVR models can tune an extra parameter, the ϵ insensitivity used in their losses, having thus more flexibility.

6. Advanced architectures

In this Section we will compare the hinge and cross entropy losses on three large image datasets using more advanced DNN architectures. More precisely, we will first apply two class versions of the well known LeNet-5 convolutional neural network [16] on both the MNIST and MNIST Fashion problems. While quite simple for today's deep proposals, LeNet-5 has still a fairly complex structure, depicted for instance in Fig. 2 of [16]. Next, we will consider a more advanced Residual Network architecture, ResNet32 V1, on the CIFAR 10 image classification problem. The ResNet structure is still more complicated, as it can be seen, for instance, in Fig. 3 of [17].

Basic dataset information is given in Table 8. GSVC models are very costly to hyperparametrize now, but for illustration purposes we will also give results for LIBLINEAR, the ad-hoc implementation for linear SVMs. While in general not competitive with GSVCs, this is less so for high dimensional problems, as the ones under consideration here. While all of them are 10 class problems, we will consider for each datasets ten 2-class problems where each single class has to be distinguished from the remaining nine classes.

6.1. The MNIST problem

MNIST is by now a classical character recognition problem that has been widely used as a DNN benchmark, with many architectures being proposed over the years. According to Rodrigo Benenson's web page [22], the state of the art networks as of 2016 achieve a test error of 0.21% [23]. The error rate when we apply the LeNet-5 implementation on the full MNIST 10 class problem is of 1.11%; a 0.95% error rate is reported in

Table 9

Accuracy comparisons of DNNC, DSVC and LibLinear for LeNet5 nets on MNIST (left: train vs test, right: test vs train; larger accuracies in bold face)

	Train vs test			Test vs train		
	DNNC	DSVC	LIBLIN	DNNC	DSVC	LIBLIN
0	99.88	99.88	99.09	99.76	99.70	98.59
1	99.91	99.87	99.44	99.67	99.70	98.37
2	99.82	99.79	98.12	99.50	99.40	97.27
3	99.87	99.79	97.91	99.48	99.57	96.78
4	99.87	99.77	98.22	99.52	99.55	97.52
5	99.77	99.71	97.93	99.55	99.48	96.60
6	99.72	99.82	98.69	99.67	99.51	98.14
7	99.62	99.63	98.28	99.45	99.45	97.81
8	99.82	99.77	96.23	99.16	99.38	95.14
9	99.63	99.56	96.57	99.47	99.29	95.47
Ave	99.79	99.76	98.05	99.52	99.50	97.17

Y. LeCun web page [24] for LeNet-5. (Notice that here and following experiments different executions may result in slight random changes in accuracy values). In our experiments we will use the MNIST dataset available in [24], which has 60,000 training and 10,000 test patterns given as 28×28 gray scale images. We will pad them on their boundaries with zeros to work with 32×32 images for better working with convolutional layers.

Our LeNet-5 implementation is fairly standard, with two convolutional layers with 3×3 filters and 6 and 16 output channels respectively, each followed by two dimensional averaging pooling. These are followed by a first dense layer with 120 units and a second one with 84 units, both with ReLU activations. For our two class problems we will replace the 10 unit, softmax outputs of LeNet-5 for the standard 10 class MNIST problem with single unit sigmoidal and linear outputs for the binary cross entropy and hinge losses, respectively. We will use Tikhonov regularization in the dense and output layers, with different penalties α_S and α_R for the output and dense weights respectively. The corresponding values are $\{0, 10^{-8}, 10^{-6}, 10^{-4}, 10^{-3}\}$ for both the α_R and α_S .

We report test accuracies under two different scenarios, a first one where we hyperparametrize and train the networks on the training set and evaluate them on the test subset, and a more difficult one, where we switch the datasets, hyperparametrizing and training now on the 10,000 pattern test set and testing the resulting models on the 50,000 patterns of the train datasets. We also use here f_1 score for CV hyperparametrization; notice that now the 2-class problem are fairly unbalanced, with the positive classes having 9 times less patterns than the negative ones. Test results are given in Table 9 for the first scenario (left) and for the second one (right); the

table also shows mean accuracies. For easier reading we put in bold face largest accuracies for each problem. This has only an illustrative value but clearly show that LIBLINEAR models fall behind. To have a more meaningful comparison we have performed a Wilcoxon test over the DNNC and DSVC accuracies along the lines detailed for the classification problems in Section 5; sample size is now 20 (10 train vs test and 10 test vs train problems). The resulting p value is 0.14, slightly above the rejection value for the same distribution hypothesis at the 0.1 level and, hence, we cannot guarantee that the DNNC and DSVC accuracy distributions are different.

6.2. The MNIST fashion problem

We will use the same LeNet-5 network on the MNIST Fashion dataset [25]. It also contains 60,000 training and 10,000 test patterns in 10 classes of clothing products (trousers, pullovers, ...) given as 28×28 gray scale images, which we pad again to size 32×32 . We have downloaded the dataset from the Zalando Research github page [26]. According to that page, the state of the art accuracy is 96.7%, achieved using wide residual networks; the test accuracy for the entire 10 class problem of our LeNet-5 network without any hyperparameterization is 90.92%.

We have performed over the MNIST Fashion dataset the same two-class experiments done for standard MNIST, hyperparameterizing again separately the Tikhonov regularization penalties of the dense hidden layer weights and of the output weights; we have used the same α_S and α_R than for MNIST. The 2-class problems considered are again fairly imbalanced and, thus, we also use the $f1$ score for validation. Accuracy test results are given now in Table 10 for the train vs test problem (left) and for the test vs train one (right); the table also shows mean accuracies. Again, for easier reading we put in bold face largest accuracies for each problem; clearly the table shows that LIBLINEAR models also fall behind. For a more meaningful comparison we have performed a Wilcoxon test along the previous lines over the DNNC and DSVC accuracies; sample size is again 20. The resulting p value is 0.48, which implies we cannot reject the same accuracy distribution hypothesis.

6.3. The CIFAR 10 problem

Our last experiment with advanced deep architectures will deal with the CIFAR-10 dataset. It contains 60,000

Table 10

Accuracy comparisons of DNNC, DSVC and LibLinear for LeNet5 nets on MNIST fashion (left: train vs test, right: test vs train; larger accuracies in bold face)

	Train vs test			Test vs train		
	DNNC	DSVC	LIBLIN	DNNC	DSVC	LIBLIN
0	97.14	97.04	95.89	96.02	95.99	95.58
1	99.69	99.73	99.36	99.60	99.51	99.39
2	96.86	97.26	94.22	96.09	95.71	94.21
3	98.31	98.19	96.47	97.43	97.62	96.66
4	97.34	97.18	94.80	95.84	95.93	94.61
5	99.74	99.72	98.46	99.31	99.24	97.44
6	94.62	95.25	92.63	93.92	93.50	92.51
7	99.33	99.31	98.15	98.72	98.92	97.59
8	99.75	99.69	98.59	99.52	99.33	98.38
9	99.44	99.38	98.67	99.12	99.12	98.23
Ave	98.22	98.28	96.72	97.56	97.49	96.46

32×32 RGB color images distributed in 10 classes (airplane, automobile, bird, ...), of which 50,000 are for training and the remaining 10,000 for test; each class has 6000 images per class. While the training subset is further divided into five training batches, we won't use them.

We shall use the ResNet32 V1 implementation in [27] of the deep residual networks proposed in Subsection 4.2 of [17]. The network has three stacks of five residual blocks, each with two convolution layers plus a residual connection. The convolutional layers at each stack result in 16, 32 and 64 output channels respectively. Originally batch normalization is applied but no regularization takes place; moreover, the outputs of the last convolutional layer are flattened and fed into ten softmax model outputs. This results in a total of about 468,000 trainable weights. In our case we just feed these flattened outputs into either a sigmoid output for the DNNC model or a linear one for the DSVC one. We point out that training here is quite costly, with a single epoch taking about 3 minutes. Because of this we have performed a limited hyperparameterization in our CIFAR 10 experiments, considering only α_S values in $\{0, 10^{-12}, 10^{-10}, 10^{-8}\}$.

The test accuracy reported in [17] is 92.49% using data augmentation; the best accuracy in [22] is 96.53%. With our implementation we have achieved a 85.33% accuracy after 200 epochs without data augmentation and 92.04% with it. Test results are given now in Table 11 for the train vs test problem (left) and for the test vs train one (right); the table also shows mean accuracies. Again, for easier reading we put in bold face largest accuracies for each problem. LIBLINEAR results are quite bad, as it seems to assign all patterns to the majority class. Applying a Wilcoxon signed rank test along the previous lines yields a p value of 0.41;

Table 11

Accuracy comparisons of DNNC, DSVC and LibLinear for ResNet32 V1 nets on CIFAR 10 (left: train vs test, right: test vs train; larger accuracies in bold face)

	Train vs test			Test vs train		
	DNNC	DSVC	LIBLIN	DNNC	DSVC	LIBLIN
0	96.46	96.00	90.11	93.52	93.41	90.50
1	98.42	98.43	90.92	96.70	96.76	90.68
2	94.36	94.04	90.00	90.49	91.15	90.00
3	92.48	90.00	90.00	90.10	90.10	90.00
4	96.03	96.10	90.00	89.93	91.23	90.00
5	94.19	94.94	90.00	90.78	92.19	90.00
6	96.66	96.48	90.01	93.92	93.61	90.00
7	97.20	96.92	90.64	93.71	94.34	90.58
8	97.75	97.53	90.56	95.76	96.08	90.00
9	97.55	97.76	90.37	95.12	95.73	90.00
Ave	96.11	95.82	90.26	93.00	93.46	90.18

Table 12

Summary of Wilcoxon's signed ranked test results for the accuracies in the MNIST, MNIST Fashion and CIFAR 10 problems, as well as for all these problems

	Mnist	Fashion	Cifar	All
Sample size	20	20	20	60
Wilcox. test statistic	51.50	77.50	74.50	735.00
p _value	0.14	0.48	0.41	0.61

again, this implies we cannot reject the same distribution hypothesis for accuracies.

Finally, and as a summary of the previous results, we give in Table 12 the sample sizes, the values of the Wilcoxon test statistics (i.e., the values returned by the Wilcoxon test algorithm) and the p values corresponding to these statistics, for the MNIST, MNIST Fashion and CIFAR 10 problems; we also give these values when considering together the DNNC and DSVC accuracies for the 60 problems considered in this Section. Contrary to what was discussed for classification problems in Section 5, now all the train and test datasets are different. As it can be seen, all the p values are above the 0.05 level, which implies that in no case we can reject the null hypothesis of the accuracy distributions being the same; this justifies the conclusion of DNNC and DSVC models perform similarly in these two-class classification problems.

7. Conclusions and further work

We have proposed deep DSVC and DSVR models which jointly learn an optimal last hidden layer representation and a maximum margin hyperplane acting upon it. Our first motivation is that Gaussian SVMs are not time competitive over large datasets. A second motivation is that the mild non differentiability of the

hinge and ϵ -insensitive losses used in SVM models can be automatically handled by modern DNN packages.

After reviewing the results in [15], we have compared our proposals against standard DNN cross entropy classifiers or squared error regressors. We do so because the counterparts in large problems of our deep SVM proposals are precisely standard DNNs. Our experiments tentatively show that DNNC and DSVC models perform similarly in classification problems, as we cannot statistically reject the null hypothesis of equal DNNC and DSVC accuracies over two-class problems (with the caveats mentioned in Section 5). The situation is different, however, in regression problems. In principle, results should be now influenced by the scoring function used to hyperparametrize regularization penalties. Indeed, when using the mean absolute error (MAE) for this, we can reject in favor of our DSVR models the null hypothesis of the DSVR and DNNR error distribution being the same. On the other hand, one would expect a better DNNR performance when using mean squared error (MSE) CV scoring; however, this turns out not to be so and DSVR models have also a light advantage here. We point to the extra ϵ DSVR hyperparameter as a possible reason for this.

As lines for further work, we first mention that large margins for DNNs have been studied in [28–31], in some cases using sigmoid/softmax outputs and binary or categorical cross entropies. On the other hand, and as mentioned, Deep SVMs naturally maximize the margins achieved in the last hidden layer. It is thus natural to compare both margins for classification problems. A second research line has to do with the kernels induced by Deep SVMs on the last hidden layer, as there has recently been a flurry of activity on possible kernel structures on the last hidden layer of highly overparametrized DNNs [32–35]). It is thus natural to study the possible relationship between these kernels and those induced by Deep SVMs on the rather wide last hidden layers considered here. Finally, one of the main advantages of the current DNN paradigms is the great flexibility they allow to define neural architectures (see for instance [36–41]), which results in their wide application [42–47]; thus, a natural question is how to take advantage of losses such as the hinge or ϵ -insensitive used here, to enhance the performance of other neural models. We are currently considering these and other related questions.

Acknowledgments

With partial support from Spain's grants TIN2016-76406-P. Work supported also by the UAM-ADIC Chair

for Data Science and Machine Learning. We gratefully acknowledge the use of the facilities of Centro de Computación Científica (CCC) at UAM.

References

- [1] Schölkopf B, Smola AJ. Learning with Kernels: support vector machines, regularization, optimization, and beyond. Adaptive computation and machine learning series. MIT Press; 2002.
- [2] Bordes A, Ertekin S, Weston J, Bottou L. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*. 2005; 6: 1579–1619.
- [3] Claesen M, Smet FD, Suykens JAK, Moor BD. EnsembleSVM: a library for ensemble learning using support vector machines. *Journal of Machine Learning Research*. 2014; 15(1): 141–145.
- [4] Joachims T, Yu CJ. Sparse kernel SVMs via cutting-plane training. *Machine Learning*. 2009; 76(2–3): 179–193.
- [5] Wen Z, Shi J, Li Q, He B, Chen J. ThunderSVM: a fast SVM library on GPUs and CPUs. *Journal of Machine Learning Research*. 2018; 19(21): 1–5.
- [6] Yu H, Hsieh C, Chang K, Lin C. Large linear classification when data cannot fit in memory. *TKDD*. 2012; 5(4): 23:1–23:23.
- [7] Hsieh C, Chang K, Lin C, Keerthi SS, Sundararajan S. A dual coordinate descent method for large-scale linear SVM. In: *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008)*, Helsinki, Finland, June 5–9, 2008; 2008. pp. 408–415.
- [8] Shalev-Shwartz S, Singer Y, Srebro N, Cotter A. Pegasos: primal estimated sub-gradient solver for SVM. *Math Program*. 2011; 127(1): 3–30.
- [9] Steinwart I, Hush DR, Scovel C. Training SVMs without offset. *J Mach Learn Res*. 2011; 12: 141–202.
- [10] Tang Y. Deep Learning using Support Vector Machines. *CoRR*. 2013; abs/1306.0239. Available from: <http://arxiv.org/abs/1306.0239>.
- [11] Zhang S, Liu C, Yao K, Gong Y. Deep neural support vector machines for speech recognition. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19–24, 2015*; 2015. pp. 4275–4279.
- [12] Chollet F. Keras: Deep Learning library for Theano and TensorFlow; 2015. Available from: <https://github.com/fchollet/keras>.
- [13] Rudin C, Carlson D. The Secrets of Machine Learning: Ten Things You Wish You Had Known Earlier to be More Effective at Data Analysis. *CoRR*. 2019; abs/1906.01998. Available from: <http://arxiv.org/abs/1906.01998>.
- [14] Angelino E, Larus-Stone N, Alabi D, Seltzer M, Rudin C. Learning certifiably optimal rule lists for categorical data. *J Mach Learn Res*. 2017; 18: 234:1–234:78.
- [15] Díaz-Vico D, Prada J, Omari A, Dorronsoro JR. Deep Support Vector Classification and Regression. In: *From Bioinspired Systems and Biomedical Applications to Machine Learning – 8th International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2019, Almería, Spain, June 3–7, 2019, Proceedings, Part II*; 2019. pp. 33–43.
- [16] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 1998; 86: 2278–2323.
- [17] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition*; 2016. pp. 770–778.
- [18] Lázaro JL, Dorronsoro JR. Simple proof of convergence of the SMO algorithm for different SVM variants. *IEEE Trans Neural Netw Learning Syst*. 2012; 23(7): 1142–1147.
- [19] Manton JH, Amblard P. A primer on reproducing kernel hilbert spaces. *Foundations and Trends in Signal Processing*. 2015; 8(1–2): 1–126.
- [20] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings*; 2015. Available from: <http://arxiv.org/abs/1412.6980>.
- [21] Chang CC, Lin CJ. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*. 2011; 2(3): 27:1–27:27. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [22] Benenson R. What is the Class of This Image? Available from: http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html.
- [23] Wan L, Zeiler M, Zhang S, LeCun Y, Fergus R. Regularization of Neural Networks Using Dropconnect. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning, Vol. 28*; 2013. pp. III–1058–III–1066.
- [24] LeCun Y. The MNIST Database of Handwritten Digits. Available from: <http://yann.lecun.com/exdb/mnist/>.
- [25] Xiao H, Rasul K, Vollgraf R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR*. 2017; abs/1708.07747.
- [26] Fashion-MNIST Z. Available from: <https://github.com/zalandoresearch/fashion-mnist>.
- [27] Keras Examples: CIFAR-10 ResNet. Available from: https://keras.io/examples/cifar10_resnet/.
- [28] Sun S, Chen W, Wang L, Liu TY. Large Margin Deep Neural Networks: Theory and Algorithms. *ArXiv*. 2015; abs/1506.05232.
- [29] Liu W, Wen Y, Yu Z, Yang M. Large-Margin Softmax Loss for Convolutional Neural Networks. In: *Proceedings of The 33rd International Conference on Machine Learning, Vol. 48 of Proceedings of Machine Learning Research*; 2016. pp. 507–516.
- [30] Liang X, Wang X, Lei Z, Liao S, Li SZ. Soft-Margin Softmax for Deep Classification. In: Liu D, Xie S, Li Y, Zhao D, El-Alfy ESM, eds. *ICONIP. Neural Information Processing*. Springer International Publishing; 2017. pp. 413–421.
- [31] Elsayed G, Krishnan D, Mobahi H, Regan K, Bengio S. Large Margin Deep Networks for Classification. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, eds. *Advances in Neural Information Processing Systems 31*; 2018. pp. 842–852.
- [32] Jacot A, Gabriel F, Hongler C. Neural Tangent Kernel: Convergence and Generalization in Neural Networks. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, eds. *Advances in Neural Information Processing Systems 31*; 2018. pp. 8571–8580.
- [33] Arora S, Du SS, Hu W, Li Z, Salakhutdinov R, Wang R. On Exact Computation with an Infinitely Wide Neural Net. *CoRR*. 2019; abs/1904.11955.
- [34] Zhang C, Bengio S, Hardt M, Recht B, Vinyals O. Understanding deep learning requires rethinking generalization. *ArXiv*. 2016; abs/1611.03530.
- [35] Olson M, Wyner A, Berk R. Modern Neural Networks Generalize on Small Data Sets. In: Bengio S, Wallach H, Larochelle

- H, Grauman K, Cesa-Bianchi N, Garnett R, eds. *Advances in Neural Information Processing Systems* 31; 2018. pp. 3619–3628.
- [36] Ahmadlou M, Adeli H. Enhanced probabilistic neural network with local decision circles: a robust classifier. *Integrated Computer-Aided Engineering*. 2010 8; 17: 197–210.
- [37] Rafiei MH, Adeli H. A new neural dynamic classification algorithm. *IEEE Trans Neural Networks Learn Syst*. 2017; 28(12): 3074–3083.
- [38] Soares AM, Fernandes BJT, Filho CJAB. Structured pyramidal neural networks. *Int J Neural Syst*. 2018; 28(5): 1750021:1–1750021:22.
- [39] Antoniadis A, Spyrou L, Martín-Lopez D, Valentín A, Alarcón G, Sanei S, et al. Deep neural architectures for mapping scalp to intracranial EEG. *International Journal of Neural Systems*. 2018; 28(8): 1850009:1–1850009:15.
- [40] Wu T, Bîlbîe FD, Paun A, Pan L, Neri F. Simplified and yet Turing universal spiking neural p systems with communication on request. *International Journal of Neural Systems*. 2018; 28(8): 1850013:1–1850013:19.
- [41] Pupo OGR, Ventura S. Performing multi-target regression via a parameter sharing-based deep network. *International Journal of Neural Systems*. 2019; 29(9): 1950014:1–1950014:22.
- [42] Nguyena T, Kashania A, Bordasb S, Ngo T. Deep neural network with higher order neuron for the prediction of foamed concrete strength. *Computer-Aided Civil and Infrastructure Engineering*. 2019; 34(5): 316–332.
- [43] Liang X. Image-based post-disaster inspection of reinforced concrete bridge systems using deep learning with Bayesian optimization. *Computer-Aided Civil and Infrastructure Engineering*. 2019; 34(5): 415–430.
- [44] Maeda K, Ogawa T, Haseyama M, Takahashi S. Convolutional sparse coding-based deep random vector functional link network for distress classification of road structures. *Computer-Aided Civil and Infrastructure Engineering*. 2019; 34(8): 654–676.
- [45] Gao Y, Kong B, Mosalam KM. Deep leaf-bootstrapping generative adversarial network for structural image data augmentation. *Computer-Aided Civil and Infrastructure Engineering*. 2019; 34(9): 755–773.
- [46] Wu RT, Ankush Singla A, Jahanshahi MR, Bertino E, Ko BJ, Verma D. Pruning deep convolutional neural networks for efficient edge computing in condition assessment of civil infrastructures. *Computer-Aided Civil and Infrastructure Engineering*. 2019; 34(9): 774–789.
- [47] Zhang Y, Cheng T, Ren Y. A graph deep learning method for short-term traffic forecasting on large road networks. *Computer-Aided Civil and Infrastructure Engineering*. 2019; 34(10): 431–449.



Deep Support Vector Classification and Regression

David Díaz-Vico^{1,2(✉)}, Jesús Prada¹, Adil Omari³, and José R. Dorronsoro^{1,2}

¹ Dpto. Ing. Informática, Universidad Autónoma de Madrid, Madrid, Spain
david.diazv@estudiante.uam.es

² Instituto de Ingeniería del Conocimiento,
Universidad Autónoma de Madrid, Madrid, Spain

³ Signal Theory and Communications Department,
Universidad Carlos III, Madrid, Spain

Abstract. Support Vector Machines, SVM, are one of the most popular machine learning models for supervised problems and have proved to achieve great performance in a wide broad of predicting tasks. However, they can suffer from scalability issues when working with large sample sizes, a common situation in the big data era. On the other hand, Deep Neural Networks (DNNs) can handle large datasets with greater ease and in this paper we propose Deep SVM models that combine the highly non-linear feature processing of DNNs with SVM loss functions. As we will show, these models can achieve performances similar to those of standard SVM while having a greater sample scalability.

1 Introduction

Support Vector Machines (SVM; [17]) are one of the state of the art methods for supervised classification and regression and, as such, widely used. One key fact for this is their ability to work implicitly with kernels such as the Gaussian one, which map the initial features to a possibly infinite dimensional reproducing kernel Hilbert space. But, on the other hand, this capability also hinders their ability to cope with large datasets, as the handling of the kernel matrix becomes too costly or, sometimes, unfeasible and, even if a model is finally built, the so called “kernelization curse”, i.e., the fact that the number of Support Vectors grows linearly with sample size, implies that the model may be too costly in memory or time to exploit.

Many proposals have appeared in the literature to overcome these problems, usually for Support Vector classification (SVC) problems. Among them we can mention the incremental learning of SVMs [1], ensemble learning of SVMs [6] or cutting planes [15] but, nevertheless, it can be said that, unless substantial hardware resources are committed, current kernel SVM training methods are not competitive for datasets with more than about 100,000 patterns.

A. Omari—Currently at Telefónica.

These problems are largely mitigated when working with linear SVMs [21] for which efficient algorithms, such as Pegasos [19] or dual coordinate descent [14], exist. However, efficient linear SVM models can only be expected when the original features have very large dimension so that projections are no longer needed. When sample dimension is just moderate, linear SVM models are usually less powerful than their Gaussian counterparts.

In principle, the main difficulty when working with SVMs is the non-differentiable nature of the SVC hinge loss that, at first sight, forces a dual problem to be solved. However, its non-differentiability is rather mild and subgradient descent is used, for instance, in the Pegasos algorithm. In fact, the SVM losses have the same non-differentiable behavior of the ReLU loss [11] routinely used in Deep Neural Networks (DNN) to compute gradients by backpropagation and which are easily handled automatically by DNN backends such as TensorFlow [12]. In fact, the hinge loss is already predefined in the Keras wrapper [5] for TensorFlow. Also, and although not predefined in Keras, the ϵ -insensitive loss used in Support Vector regression (SVR) has the same nondifferentiable behavior.

This suggests to consider an alternative to non-linear SVMs by means of DNNs for which the standard cross entropy for classification or squared error for regression are replaced by the hinge and ϵ -insensitive losses respectively. This approach has been already applied in [20], where it is compared with standard softmax DNNs on classification problems, and by [22] in speech recognition. Here we will also extend it to Deep Support Vector Regression models and we will compare these Deep SV Classifiers and Regressors with their standard Gaussian SVC and SVR counterparts in a number of medium to large classification and regression problems.

The paper is organized as follows. In Sect. 2 we will briefly review standard kernel-based SVC and SVR models while their deep counterparts are explained in Sect. 3. Numerical experiments are presented in Sect. 4 and the paper ends with a discussion, conclusions and pointers to further work.

2 Support Vector Machines

Given a sample $S = \{(x^p, y^p), p = 1, \dots, N, y^p = \pm 1\}$, if we have a linear model $\xi = f(x) = w \cdot x + b$ and h denotes the hinge loss

$$h(y, f(x)) = \max\{0, 1 - yf(x)\} = \max\{0, 1 - y\xi\} = h(y, \xi),$$

the optimization problem to be solved for a linear SVC is

$$\begin{aligned} & \arg \min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_1^N h(y^p, w \cdot x^p + b) \\ & \equiv \arg \min_{w,b} \left(\frac{1}{2CN} \|w\|^2 + \frac{1}{N} \sum_1^N h(y^p, w \cdot x^p + b) \right) \\ & \equiv \arg \min_{w,b} \left(\frac{1}{N} \sum_1^N h(y^p, w \cdot x^p + b) + \alpha \|w\|^2 \right) \end{aligned} \quad (1)$$

with $\alpha = \frac{1}{2CN}$. In general, SVCs are usually built solving the dual problem of (1), namely

$$\max_{\alpha_i} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j x_i^T x_j \quad (2)$$

subject to $0 \leq \alpha_i \leq C$, $i = 1, \dots, N$, $\sum_{i=1}^N \alpha_i y_i = 0$.

Notice that problems (1) and (2) only involve dot products; this is also the case of the final SVM model. As a consequence, all can be written as just done but replacing x with a kernel-related projection $\Phi(x)$. The map Φ can be implicit, as is the case with standard kernel SVCs, where the dot product only enters through some kernel, or it can be an explicit one, as it will be the case here, where $\Phi(x) = F(x, \mathcal{W}_h)$ will be the vector of last hidden layer values of a MLP with input to last hidden layer weights \mathcal{W}_h . Moreover, if the kernel representation $\Phi(x^p)$ is known explicitly, the primal problem (1) can be directly solved, as done by the Pegasos algorithm [19] (see also [3], where the primal problem is solved for the squared hinge loss). We will also exploit the explicit kernel knowledge in the Deep SVC approach that we describe in the next section.

Given now a sample $S = \{(x^p, y^p), p = 1, \dots, N\}$ with the y^p here being numerical targets, and a linear model $\xi = f(x) = w \cdot x + b$, the ϵ -insensitive loss, $\epsilon > 0$, would now be

$$\ell_\epsilon(y, f(x)) = \ell_\epsilon(y, \xi) = \max\{0, |\xi| - \epsilon\}.$$

The optimization problem to be solved now for linear SV Regression (SVR) is

$$\arg \min_{w, b} \frac{1}{N} \sum_{i=1}^N \ell_\epsilon(y^p, w \cdot x^p + b) + \alpha \|w\|^2. \quad (3)$$

As it was the case in SVC, the ϵ -insensitive loss is convex but only piecewise differentiable and because of this, here again, the task solved in practice is the following dual problem derived through the Lagrangian formalism

$$\begin{aligned} \max_{\alpha_i, \alpha_i^*} & \sum_{i=1}^N y_i (\alpha_i^* - \alpha_i) - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N (\alpha_i^* - \alpha_i) (\alpha_j^* - \alpha_j) x_i \cdot x_j - \\ & \epsilon \sum_{i=1}^N (\alpha_i^* + \alpha_i) \end{aligned} \quad (4)$$

subject now to $0 \leq \alpha_i, \alpha_i^* \leq \frac{1}{2\alpha N}$, $i = 1, \dots, N$, $\sum_{i=1}^N (\alpha_i^* - \alpha_i) = 0$.

The kernel trick can also be applied here and, if it is known explicitly, gradient or subgradient descent can be used. We describe next how to exploit this for Deep SVR.

3 Deep Support Vector Machines

3.1 Deep Learning

Artificial Neural Networks (ANNs) can be seen as an extension of the classical linear and logistic regression models. They can generate an arbitrarily good latent representation of the data [8, 13] that can be efficiently used to build powerful models. Starting with the multi layer perceptrons (MLPs), whose basic theory was already well established in the 80s, and the backpropagation algorithm for gradient computation during ANN training, they can be considered as the first example of modern machine learning algorithms that could be applied to both regression and classification problems with minimal conceptual variations. However, technical difficulties, essentially due to knowledge gaps about their training plus the lack of computing power at the time, led to their relative decline in the late 90's and the rise of competing methods, particularly SVMs, for classification and regression.

But about 2010, the wide availability of powerful computing facilities, advances on the theoretical underpinnings of MLPs, many refinement of their training procedures and a better understanding of the difficulties related to many layered architectures [10] have produced a spectacular expansion under the deep neural network (DNN) paradigm. To all this we can add the appearance of several development frameworks such as TensorFlow [12], CNTK [18], MXNET [4] or Torch [7], as well as wrappers for them such as Keras [5], that have allowed the practitioners to experiment with different architectures, non-differentiable activations and, even, non-differentiable loss functions. Last, but not least, the iterative nature of backpropagation and its linear cost growth with respect to sample size imply that DNNs are much less affected from the bad scalability of kernel methods, and can be applied with relative ease to datasets with hundreds of thousands of patterns. As mentioned, mildly non-differentiable functions such as ReLUs or the hinge loss, can be handled within the DNN framework and are relatively simple to incorporate into DNN tools. We discuss next how the hinge and ϵ -insensitive loss can be introduced in a DNN set up.

3.2 Deep SVC and SVR

Consider a more or less standard DNN architecture where an input layer is followed by a number of hidden layers that can be either convolutional or fully connected or of any other type, and, finally, linear output activations. The transformation of such a network can be thus written as

$$f(x, w, b, \mathcal{W}_h) = w \cdot F(x, \mathcal{W}_h) + b = w \cdot \Phi(x) + b,$$

where \mathcal{W}_h denotes the set of weights and biases up to the last hidden layer, w, b denote the linear weights and bias acting on this last hidden layer, and $F(x, \mathcal{W}_h)$ the last hidden layer outputs. We also denote these outputs as $\Phi(x)$ when we want to emphasize a kernel perspective.

Considering first SVCs, the hinge loss is differentiable, as mentioned, everywhere except at $\xi = 0$; the same is true for the ReLU activations commonly used in deep networks. As it is the case of ReLUs, the hinge loss can be implemented using the primitives of backends such as TensorFlow, and loss gradients are derived automatically when network models are compiled. Here, the optimal weights are to be obtained minimizing the following regularized cost

$$J(w, b, \mathcal{W}_h) = \frac{1}{N} \sum_1^N h(y^p, w \cdot F(x^p, \mathcal{W}_h) + b) + \alpha_S \|w\|^2 + \alpha_H \mathcal{R}(\mathcal{W}_h); \quad (5)$$

here $\|w\|^2$ is the squared norm of the linear output weights and $\mathcal{R}(\mathcal{W}_h)$ can be any regularizer function acting on the weights of the hidden layers. The preceding formulation points for instance to the regularization given by the Frobenius norm of the weights of fully connected or convolutional layers. Notice that, as explained below, we would keep in principle different weight penalties α_S and α_H for the linear output weights and for the hidden layer ones; if dropout was to be used, α_H would be the dropout probability.

The minimization of (5) can be achieved by standard DNN solvers such as stochastic gradient descent, Adam [16], Adagrad [9] or others. The regularization parameters α_S , α_H are to be selected by some form of cross validation (as well as the DNN architecture, if desired). If we work with a predefined DNN architecture, we will have to optimize here two hyperparameters, α_S , α_H , the same number than in Gaussian SVC, where the C penalty and γ kernel width have to be adjusted.

Notice that after optimization, the w^* , b^* weights correspond to a linear SVC acting on the hidden layer outputs $F(x^p, \mathcal{W}_h^*)$, for they must solve

$$\arg \min_{w,b} \frac{1}{N} \sum_1^N h(y^p, w \cdot z^p + b) + \alpha_S \|w\|^2 \quad (6)$$

with $z^p = F(x^p, \mathcal{W}_h^*)$. This is just the problem solved by a linear SVC on the z^p patterns; in particular, its margin is given by $\frac{1}{\|w^*\|}$. Since $\|w^*\|$ will be controlled by α_S while the other layers have different goals, this seems to point to the convenience of working with separate penalties α_S and α_H . Observe also that although the network's outputs $\xi = w^* \cdot F(x, \mathcal{W}_h^*) + b^*$ are continuous real numbers, the class prediction \hat{y} is obtained as $\hat{y} = \text{sign}(\xi)$. Therefore, it is straightforward to obtain discrete classification scores such as accuracy, precision or recall, but some further work would be needed to achieve a more fine-grained prediction that makes possible to compute, for instance, ROC curves or their AUC. Possible ways of doing so would be to consider moving classification thresholds or to have a `predict_proba` method where the raw outputs of the Deep SVC were calibrated in some way.

Turning our attention to SVR, we again consider a more or less standard DNN architecture with linear outputs that defines a nonlinear transformation $f(x, w, b, \mathcal{W}_h) = w \cdot F(x, \mathcal{W}_h) + b = w \cdot \Phi(x) + b$ and a loss

$$J_\epsilon(w, b, \mathcal{W}_h) = \frac{1}{N} \sum_1^N \ell_\epsilon(y^p, w \cdot F(x^p, \mathcal{W}_h) + b) + \alpha_S \|w\|^2 + \alpha_H \mathcal{R}(\mathcal{W}_h) \quad (7)$$

where the regularization parameters α_S , α_H (and, if desired, the DNN architecture) are selected again by some form of cross validation. The optimal w^* , b^* will now coincide with those obtained by a linear SVR model acting on the last hidden layer outputs $z^p = F(x^p, \mathcal{W}_h^*)$. In Deep SVR we will have to optimize now three hyperparameters, α_S , α_H and the ϵ -insensitivity value, again the same number than in Gaussian SVRs.

In summary, essentially the same network structure is used for Deep SVC and SVR and the only differences are in the targets (± 1 for classification, real numbers for regression) and the loss used in each situation. In principle, we could expect classification accuracies or regression errors to be more or less the same when standard Gaussian or Deep SVC/SVR models are used, although the greater flexibility of the Deep SVC/SVR architectures should also result in a better performance in some problems, such as for instance, those dealing with images. In any case, the main advantage of the Deep SVC/SVR networks is likely to be manageable training times and, perhaps, more importantly, the ability to build models over large samples. Current kernel SV models have substantial difficulties with sample sizes above 100,000 and sizes above 500,000 are often outside their present reach unless really substantial computing resources are available. On the other hand, Deep SV models will be costly but still possible to be trained on those sample sizes. Moreover, prediction times will certainly be much faster for Deep SV models than for kernel based ones.

4 Experiments

4.1 Classification Experiments

We compare the performance of Kernel and Deep SVC over the datasets `a4a`, `a8a`, `australian`, `cod-rna`, `diabetes`, `german.numer`, `ijcnn1`, `w7a` and `w8a` from the LibSVM repository [2]. Their training and, when available, test sample sizes and dimensions are given in Table 1. In the Deep SVC (DSVC) models we will use ReLU hidden layer activations and Adam over minibatches as the loss optimizer. To lighten the hyper-parameterization costs we will use the default values in the Keras implementation of Adam for the initial learning rate (0.001) and the β_1 (0.9) and β_2 (0.999) parameters. We shall also use in all cases a minibatch size of 200. Notice that network architecture, i.e., the number of hidden layers and of their units, could also be considered as hyper-parameters to be optimized. However we will work with DSVC models with 1 to 5 hidden layers and 100 units in all layers but the last one, which will have $0.1 \times |S|$ units, with $|S|$ denoting sample size, with a lower bound of 100 and an upper bound of 1000. The rationale for this is to enlarge the dimension of the last hidden layer projections of the sample patterns, so that the linear SVC models that act on them can have a large enough representation power.

Table 1. Number of train and test patterns and dimensions in the two-class problems.

	No. patterns train	No. patterns test	Dimension
a4a	4781	27780	123
a8a	22696	9865	123
australian	690		14
cod-rna	59535	271617	8
diabetes	768		8
german.numer	1000		24
ijcnn1	49990	91701	22
w7a	24692	25057	300
w8a	49749	14951	300

Table 2. Accuracies in the two-class problems.

	SVC	DSVC1	DSVC2	DSVC3	DSVC4	DSVC5	Best DSVC
a4a	84.32 (1)	84.19 (5.5)	84.20 (4)	84.19 (5.5)	84.29 (2)	84.27 (3)	84.29
a8a	84.92 (6)	85.18 (1)	84.95 (5)	85.14 (2.5)	85.08 (4)	85.14 (2.5)	85.18
australian	85.50 (5)	85.51 (4)	87.09 (1)	86.82 (2)	86.09 (3)	85.21 (6)	87.09
cod-rna	96.58 (5)	96.66 (1)	96.62 (2)	96.61 (3)	96.60 (4)	96.45 (6)	96.66
diabetes	77.60 (1)	76.95 (4)	77.47 (2)	76.43 (5)	76.43 (6)	77.21 (3)	77.47
german.numer	76.10 (2)	75.80 (3.5)	75.20 (5)	75.10 (6)	76.60 (1)	75.80 (3.5)	76.60
ijcnn1	97.93 (6)	98.88 (4)	99.03 (2)	99.07 (1)	98.84 (5)	98.99 (3)	99.07
w7a	98.87 (1)	98.82 (5)	98.85 (2)	98.83 (3.5)	98.79 (6)	98.83 (3.5)	98.85
w8a	99.04 (2)	98.99 (5)	98.92 (6)	99.16 (1)	99.04 (3)	99.03 (4)	99.16
rank mean	3.2222	3.6667	3.2222	3.2778	3.7778	3.8333	

We thus will hyperparameterize the L_2 (or Tikhonov) regularization penalties. We will consider two different such penalties one α_S , for the output weights and another one α_H which is the same for all the hidden layer weights. In both cases we will explore 5 values evenly spaced on a log scale in the interval $[2^{-30}, 2^{10}]$ selecting the optimal one by 4-fold cross validation as described below. As customary with neural networks, the Deep SVC inputs have been normalized feature-wise to 0 mean and 1 standard deviation.

The SVC models require two hyperparameters, the C regularization term and the width γ of the Gaussian kernels $\exp(-\gamma\|x - x'\|^2)$. For C we will explore 5 values in the interval $[10^{-3}, 10^6]$. In order to select the Gaussian kernel width γ , we scale feature-wise the SVC inputs to a $[0, 1]$ range; the rationale for this is that after this normalization, we have $\|x - x'\|^2 \leq d$ with d pattern dimension. Because of this we will explore γ values of the form $\frac{2^k}{d}$, with k in the $[-3, 6]$ range.

Table 3. Number of patterns and dimensions in the regression problems.

	No. patterns	Dimension
abalone	4177	8
bodyfat	252	14
cpusmall	8192	12
housing	506	13
mg	1385	6
mpg	392	7
pyrim	74	27
space_ga	3107	6

Turning now to the cross validation (CV) procedure, when there is a separate test set, we find model hyperparameters by 4-fold stratified CV over the train set and then report the performance of the best hyperparameter model over the test set. When there is only one dataset, we use a nested two loop CV approach in order to assess model performance. More precisely, we apply stratified 4-fold CV on both loops. In the outer loop each one of the four outer folds is set apart for testing and the dataset made of the other three folds is passed to the inner loop, where again 4 fold CV is applied to determine the best hyperparameters, which are then tested on the test fold set apart. Model performance is measured as the average over these 4 test folds. Notice that while in the first case a single best parameter set is found, in the nested procedure optimal model hyperparameters may be different for each one of the outer test folds. In all cases the score used is model accuracy.

We report in Table 2 the accuracies of the Gaussian SVCs (SVC) and of Deep SVCs with 1 (DSVC1) to 5 (DSVC5) layers. For a better reading the table also gives in parenthesis the ranking of these accuracies. Similarly, we also give in the last column the accuracy of the best performing deep SVC model. This is done on a descriptive basis, as we have not performed a statistical analysis of the accuracy table, given the relatively small number of datasets considered. Gaussian SVCs give the largest accuracies on four datasets, `a4a`, `cod-rna`, `diabetes` and `w7a`. For the other five datasets the highest accuracy is achieved by a Deep SVC model. The row at the bottom of the table gives the average rankings. The better performing models appear to be SVC, DSVC2 and DSVC3. But, in any case, notice that all accuracy values are quite similar; this seems to imply that a statistical analysis of these accuracies would show them to be essentially the same. As mentioned before, the difference would lie in training and, particularly, test times, much lower for the deep models over the larger datasets.

4.2 Regression Experiments

Turning our attention to the regression problems, the datasets used to assess the performance of Kernel SVR and Deep SVR are `abalone`, `bodyfat`, `cpusmall`, `housing`, `mg`,

Table 4. MAEs in the regression problems.

	SVR	DSVR1	DSVR2	DSVR3	DSVR4	DSVR5	Best DSVR
abalone	1.48 (2)	1.49 (4)	1.49 (3)	1.50 (5)	1.48 (1)	1.51 (6)	1.48
bodyfat ($\times 100$)	0.05 (1)	0.42 (4)	0.31 (3)	0.51 (5)	0.53 (6)	0.28 (2)	0.28
cpusmall	2.13 (2)	2.21 (4)	2.12 (1)	2.19 (3)	2.31 (5)	2.40 (6)	2.12
housing	2.28 (1)	2.58 (6)	2.51 (4)	2.34 (3)	2.30 (2)	2.57 (5)	2.30
mg ($\times 100$)	9.26 (2)	9.68 (6)	9.65 (5)	9.13 (1)	9.36 (3)	9.58 (4)	9.13
mpg	1.91 (1)	2.39 (4)	2.28 (2)	2.49 (6)	2.42 (5)	2.36 (3)	2.28
pyrim ($\times 100$)	5.62 (1)	6.78 (3)	8.39 (5)	6.45 (2)	8.81 (6)	8.02 (4)	6.45
space_ga ($\times 100$)	9.67 (6)	9.14 (4)	9.22 (5)	8.63 (1)	8.70 (2)	8.86 (3)	8.63
rank mean	2	4.375	3.5	3.25	3.75	4.125	

mpg, pyrim and space_ga, again taken from the LibSVM repository [2]. Their sample sizes and dimensions are given in Table 3; notice that in this case there are no test sets.

The experimental setup is now exactly the same as in the classification setting, with the exception of the extra hyper-parameter needed for the ϵ -insensitive loss used in regression. This hyper-parameter ϵ is searched in a grid of 5 values distributed in a log scale between $2^{-10} \times std(y)$ and $2^{-1} \times std(y)$. Since there are no separate test sets, model performance in all cases is estimated by nested 4 fold CV.

We report now in Table 4 the mean absolute errors (MAEs) of the Gaussian SVRs (SVR) and of Deep SVRs with 1 (DSVR1) to 5 (DSVR5) layers. Again, the table also gives in parenthesis the ranking of these MAEs for a better reading; no statistical analysis has been performed here, again because of the relatively small number of datasets involved. We also give here the MAE of the best performing deep SVR model in the last column. Gaussian SVRs give the smallest MAE on the four smaller datasets, bodyfat, housing, mpg and pyrim, and give the second smallest MAE in three others. A Deep SVR model gives the smallest MAE for the other four datasets. Looking at the bottom row of the table, which has the average rankings, the SVR is clearly in the first place but, nevertheless, here again all MAE values are quite similar, something that a statistical test would probably confirm.

5 Conclusions and Further Work

In this paper we have shown that Deep SVC or SVR models, i.e., more or less standard DNNs with linear outputs and either hinge or ϵ -insensitive losses can give classification accuracies or mean absolute errors similar or even slightly better than those of Gaussian SVC or SVR models, but with much more manageable computational training and, particularly, test costs.

However, our experimental results open the way to more questions. A first one is which margin structure arises on the last hidden layer representations of

the sample patterns. Moreover, we have also seen that each problem seems to have its own optimal DNN architecture. Notice that Gaussian (or other kernel) SVMs also define a particular kind of architecture in terms of the kernel width hyper-parameter γ and the concrete support vectors found during training. This suggests that possible Deep SVM architectures should also be considered as hyperparameters to be adequately found. If this is done, it is likely that we would find a tie between the SVC/SVR performance and that of the best deep counterpart. Finally, when used for classification, DNNs with softmax outputs and cross entropy loss automatically yield posterior probabilities. However, this is not the case with standard SVC and, thus, neither with the deep SVMs proposed here. These and other related questions are currently under study.

Acknowledgments. With partial support from Spain’s grants TIN2016-76406-P and S2013/ICE-2845 CASI-CAM-CM. Work partially supported also by project FACIL-Ayudas Fundación BBVA a Equipos de Investigación Científica 2016, and the UAM-ADIC Chair for Data Science and Machine Learning. We also gratefully acknowledge the use of the facilities of Centro de Computación Científica (CCC) at UAM.


References

1. Bordes, A., Ertekin, S., Weston, J., Bottou, L.: Fast kernel classifiers with online and active learning. *J. Mach. Learn. Res.* **6**, 1579–1619 (2005)
2. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. *ACM Trans. Intell. Syst. Technol.* **2**(3), 27:1–27:27 (2011). <http://www.csie.ntu.edu.tw/~cjlin/libsvm>
3. Chang, K., Hsieh, C., Lin, C.: Coordinate descent method for large-scale L2-loss linear support vector machines. *J. Mach. Learn. Res.* **9**, 1369–1398 (2008)
4. Chen, T., et al.: MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR* abs/1512.01274 (2015)
5. Chollet, F.: Keras: deep learning library for Theano and TensorFlow (2015). <https://github.com/fchollet/keras>
6. Claesen, M., Smet, F.D., Suykens, J.A.K., Moor, B.D.: Ensemblesvm: a library for ensemble learning using support vector machines. *J. Mach. Learn. Res.* **15**(1), 141–145 (2014)
7. Collobert, R., Kavukcuoglu, K.: Torch7: a matlab-like environment for machine learning. In: *BigLearn, NIPS Workshop* (2011)
8. Cybenko, G.: Approximation by superpositions of a sigmoidal function. *Math. Control Sig. Syst. (MCSS)* **2**(4), 303–314 (1989)
9. Duchi, J.C., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.* **12**, 2121–2159 (2011)
10. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, pp. 249–256, May 2010
11. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, April 2011

12. Google: Tensorflow, an open source software library for machine intelligence. <https://www.tensorflow.org/>
13. Hornik, K.: Approximation capabilities of multilayer feedforward networks. *Neural Netw.* **4**(2), 251–257 (1991)
14. Hsieh, C., Chang, K., Lin, C., Keerthi, S.S., Sundararajan, S.: A dual coordinate descent method for large-scale linear SVM. In: *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008)*, Helsinki, Finland, 5–9 June 2008, pp. 408–415 (2008)
15. Joachims, T., Yu, C.J.: Sparse kernel SVMs via cutting-plane training. *Mach. Learn.* **76**(2–3), 179–193 (2009)
16. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. *CoRR* abs/1412.6980 (2014)
17. Schölkopf, B., Smola, A.J.: *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge (2002)
18. Seide, F., Agarwal, A.: CNTK: Microsoft’s open-source deep-learning toolkit. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA, 13–17 August 2016, p. 2135 (2016)
19. Shalev-Shwartz, S., Singer, Y., Srebro, N., Cotter, A.: Pegasos: primal estimated sub-gradient solver for SVM. *Math. Program.* **127**(1), 3–30 (2011)
20. Tang, Y.: Deep learning using support vector machines. *CoRR* abs/1306.0239 (2013). <http://arxiv.org/abs/1306.0239>
21. Yu, H., Hsieh, C., Chang, K., Lin, C.: Large linear classification when data cannot fit in memory. *TKDD* **5**(4), 23:1–23:23 (2012)
22. Zhang, S., Liu, C., Yao, K., Gong, Y.: Deep neural support vector machines for speech recognition. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015*, South Brisbane, Queensland, Australia, 19–24 April 2015, pp. 4275–4279 (2015)



Companion Losses for Deep Neural Networks

David Díaz-Vico²() , Angela Fernández¹, and José R. Dorronsoro^{1,2}

¹ Department of Computer Engineering, Universidad Autónoma de Madrid, Madrid, Spain

² Inst. Ing. Conocimiento, Universidad Autónoma de Madrid, Madrid, Spain

david.diaz@iic.uam.es

Abstract. Modern Deep Neuronal Network backends allow a great flexibility to define network architectures. This allows for multiple outputs with their specific losses which can make them more suitable for particular goals. In this work we shall explore this possibility for classification networks which will combine the categorical cross-entropy loss, typical of softmax probabilistic outputs, the categorical hinge loss, which extends the hinge loss standard on SVMs, and a novel Fisher loss which seeks to concentrate class members near their centroids while keeping these apart.

1 Introduction

After the seminal work of G. Hinton [8] and J. Bengio [11] and starting about 2010, Deep Neural Networks (DNNs) have exploded in terms of scientific advances, technological improvements and great successes on many applications. There are many reasons for this, but paramount among them is the great flexibility that modern DNN environments such as TensorFlow [1] or PyTorch [9] allow to define, train and exploit DNN models. Key for this are the modern tools for automatic differentiation that make possible the definition of very general network architectures and losses. For instance, this has made possible to incorporate under a DNN framework cost functions such as the hinge and ϵ -insensitive losses, with models and results that are very competitive with those of the standard Gaussian SVMs [5].

Once that more general losses are available, a natural next step is to try to combine some of them in principle independent losses within the same network, so that they can take advantage of their different goals to jointly improve on their individual achieved results. For instance, consider for two-class problems the competing cross-entropy loss, customarily used for DNN classification, with the SVM

The authors acknowledge financial support from the European Regional Development Fund and the Spanish State Research Agency of the Ministry of Economy, Industry, and Competitiveness under the project PID2019-106827GB-I00. They also thank the UAM-ADIC Chair for Data Science and Machine Learning and gratefully acknowledge the use of the facilities of Centro de Computación Científica (CCC) at UAM.

© Springer Nature Switzerland AG 2021

H. Sanjurjo González et al. (Eds.): HAIS 2021, LNAI 12886, pp. 1–12, 2021.

https://doi.org/10.1007/978-3-030-86271-8_45

hinge loss. The goal of the former is to assume a certain posterior probability and estimate a model that maximizes its sample based likelihood, while that of the latter is essentially to find a separating hyperplane with a margin as large as possible given the sample. A similar situation arises in multiclass problems, where the categorical cross-entropy with softmax outputs is used for DNN classifiers while the categorical hinge loss [3] is used for multiclass SVM-like classifiers. In these problems one or several of the losses act as the main one, while the others act as companions in the sense that they accompany the main loss towards a better model. This idea of combining losses has been applied in other areas of knowledge, specially in computer vision [10, 14], although following a different rationale.

Although we will not deal with them here, competing losses for regression problems would be the squared error of regression DNNs and the ϵ -insensitive loss used in support vector regression. Again, the underlying problem is basically the same, but the latter establishes an ϵ -wide tube around the fitted model and only penalizes errors outside the tube. This results on models more robust with respect to outliers but with the drawback of ignoring small errors, which may be important in some settings and that the squared error does not ignore. Given these different but not necessarily competing goals, it is in principle conceivable that both losses could work together towards building a model that improves on those built separately with each loss.

The goal of this work is precisely to explore these possibilities for classification problems. More precisely, we will compare the combination of the cross-entropy and hinge losses for two-class problems, and that of the categorical cross-entropy and hinge losses in multiclass ones. To these we will add a squared loss-based cost function which enforces for its inputs on each class to be concentrated near the class centroids while trying to keep these centroids apart. This approach has been proved [13] to yield linear models whose outputs can theoretically be seen to be equivalent with those provided by the classical Fisher Discriminant Analysis and that has been extended to a DNN setting in [4]. While trying to yield a classifier directly, such a loss can produce a pattern representation on the last hidden layer of a DNN which can make easier the job of a classifier acting on these representations and, hence, result in a better model.

We will work with a substantial number of classification problems and our results point out that this approach can indeed give such results. In fact, and as we will experimentally show, combining the Fisher loss with the cross-entropy or hinge ones improves on the models obtained when only single losses are used. On the other hand, the cross-entropy plus hinge combination ties at best with a single cross-entropy loss. This has to be further studied but a possible reason may be that, at least in our experiments, cross-entropy DNNs yield better results than hinge-base ones (we address reasons for this later in this paper). In summary, our contributions here are:

- The proposal of DNNs with combined losses for two- and multi-classification, which we implement as Keras [2] functional models.
- A substantial experimental work showing positive results that deserve further study.

The rest of the paper is organized as follows. We review the losses used in Sect. 2, discuss how to combine them and give some implementation details. Section 3 contains details on the datasets used, the experimental methodology and results, and a discussion and a final section offers some conclusions as well as pointers to further work. We point out that, throughout the paper, by deep networks we mean artificial neural networks that use modern techniques such as automatic differentiation, Glorot-Bengio initializations [6], ReLU activations or Adam optimizers [7], rather than they having actually deep (i.e. many layered) architectures; in fact, in our experiments we will apply all these techniques but on a single layer network with 100 units.

2 Classification Losses

Throughout this section we will work with DNN architectures that yield models acting on a pattern x with outputs $F(x, \mathcal{W})$, where \mathcal{W} denotes the set of weight matrices and bias vectors associated with the network's architecture. We will denote targets as y , which can be either $\{-1, 1\}$ for two-class problems or one-hot encoded vectors for multiclass ones. We denote the network outputs at the last hidden layer as $z = \Phi(x, \widetilde{\mathcal{W}})$, with $\widetilde{\mathcal{W}}$ the weight and bias set of all layers up to the last one. Such a z is then transformed as $Wz + B$, where W, B are either the transpose of an n_H dimensional vector w , with n_H the number of hidden units in the last hidden layer, and a scalar b , or a $K \times n_H$ matrix and a K dimensional vector in a K -class problem. These $Wz + B$ will be the final network outputs in the case of the deep SVM (or deep Fisher networks, as we describe them below); for the more standard DNN classifiers, the network output is obtained applying to them either a sigmoid or a softmax function.

In more detail, and starting with two-class problems, the usual DNN loss is the binary **cross-entropy**, given by

$$\ell_{bc}(\mathcal{W}; S) = \ell(w, b, \widetilde{\mathcal{W}}) = - \sum_p y^p (w \cdot z^p + b) + \sum_p \log(1 + e^{w \cdot z^p + b}) \quad (1)$$

where S denotes an i.i.d. sample $S = \{(x^p, y^p)\}$ with N patterns, z represent the last hidden layer outputs, w represent the weights which connects the last hidden layer with the network's output and b is the vector bias of the output. We recall that this expression is just the sample's minus log-likelihood associated to the assumption

$$P(y|x) = P(y|x; w, b, \widetilde{\mathcal{W}}) = \frac{1}{1 + e^{-y(w \cdot z + b)}} \quad (2)$$

for the posterior probability of the y class, and we assume a sigmoid network output. For general multiclass problems, the network output function is the softmax

$$F_j(x; \mathcal{W}) = \frac{e^{w_j \cdot z + b_j}}{\sum_{k=0}^{K-1} e^{w_k \cdot z + b_k}}. \quad (3)$$

Obviously then $\sum_j F_j(x; \mathcal{W}) = 1$ and we assume $P(j|x) \simeq F_j(x; \mathcal{W})$. For two-class problems this reduces to the previous output if we take $w = w_0 - w_1$. For

the loss to be used here, we assume one-hot encoded targets, i.e., the target of the k -th class is $e_k = (0, \dots, \underbrace{1}_k, \dots, 0)$; then, the probability of getting patterns x^p in class k_p (i.e., $y_{k_p}^p = 1$) within an i.i.d. sample $S = (X, Y)$ is

$$P(Y|X; \mathcal{W}) = \prod_{p=1}^N P(k_p|x^p; \mathcal{W}) = \prod_{p=1}^N \prod_{m=0}^{K-1} P(m|x^p; \mathcal{W})^{y_m^p} \simeq \prod_{p=1}^N \prod_{m=0}^{K-1} F_c(x; \mathcal{W})^{y_m^p}, \quad (4)$$

and we estimate the DNN's weights \mathcal{W} by minimizing the minus log of the approximate sample's likelihood

$$\tilde{P}(Y|X; \mathcal{W}) = \prod_{p=1}^N \prod_{m=0}^{K-1} F_m(x^p; \mathcal{W})^{y_m^p}. \quad (5)$$

That is, we will minimize the categorical cross-entropy loss

$$\ell_{cce}(\mathcal{W}) = -\log \tilde{P}(Y|X; \mathcal{W}) = -\sum_{p=1}^N \sum_{m=0}^{K-1} y_m^p \log F_m(x^p; \mathcal{W}). \quad (6)$$

Once an optimal weight set \mathcal{W}^* has been obtained, the decision function on a new x is given by $\arg \max_m F_m(x; \mathcal{W}^*)$, i.e. the class with the maximum posterior probability.

Turning our attention to two-class SVMs, the local loss is now the **hinge loss** $h(x, y) = \max\{0, 1 - yF(x, \mathcal{W})\}$; here the network has linear outputs, i.e., $F(x; \mathcal{W}) = F(x; w, b, \tilde{\mathcal{W}}) = w \cdot \Phi(x, \tilde{\mathcal{W}}) + b$. The global loss is now

$$\ell_h(\mathcal{W}; S) = \sum_p \max\{0, 1 - y^p F(x^p, \mathcal{W})\}. \quad (7)$$

There are several options to extend SVMs for multiclass problems. The usual approach in a kernel setting is to use either a one-vs-one (ovo) or a one-vs-rest (ovr) approach so that just binary kernel classifiers have to be built. Unfortunately, this cannot be directly translated to a DNN setting but there are two other ways to define multiclass local losses. The first one is due to Weston and Watkins [12] which for a pattern x in class k_x (i.e., $y_{k_x} = 1$) propose the local loss

$$\ell(x, y) = \max \left\{ 0, \sum_{m \neq k_x} (1 + F_m(x) - F_{k_x}(x)) \right\} \quad (8)$$

where we denote as $F_0(x), \dots, F_{K-1}(x)$ the network's lineal outputs. The alternative to this is the local loss proposed by Crammer and Singer [3], namely

$$\ell(x, y) = \max \left\{ 0, \max_{m \neq y} (1 + F_m(x) - F_{k_x}(x)) \right\}. \quad (9)$$

Notice that both coincide for two-class problems and, moreover, if in this case we require $w_0 = w_1 = \frac{1}{2}w$ and $b_0 = b_1 = \frac{1}{2}b$, they coincide with the local hinge loss. We will use the second one, that results in the categorical hinge global loss

$$\ell_{ch}(\mathcal{W}) = \sum_{p=1}^N \max \left\{ 0, 1 - F_{k_{x^p}}(x^p) + \max_{m \neq k_{x^p}} F_m(x) \right\}; \quad (10)$$

here $F_m(x)$ denotes the m -th component of the network's K dimensional linear output, i.e., $F_m(x) = w_m \cdot \Phi(x; \widetilde{\mathcal{W}}) + b_m$. Now, once an optimal weight set \mathcal{W}^* has been obtained, the decision function on a new x is given again by $\arg \max_m F_m(x; \mathcal{W}^*)$, although now no posterior probabilities are involved.

We will finally consider what we call the **Fisher loss**. The goal in standard Fisher Discriminant Analysis is to linearly project patterns so that they concentrate near the projected class means while these are kept apart. To achieve this, one seeks to maximize the trace criterion

$$g(A) = \text{trace}(s_T^{-1} s_B) = \text{trace} \left((A^t S_T A)^{-1} (A^t S_B A) \right), \quad (11)$$

where A is the projection matrix, S_B and S_T denote the between-class and total covariance matrices, respectively, of the sample patterns and s_B and s_T are their counterparts for the projections $z = Ax$. Solving $\nabla_A g = 0$ leads to the system $S_T^{-1} S_B A = A \Lambda$, with Λ the non-zero eigenvalues of $S_T^{-1} S_B$. For such an A we have

$$g(A) = \text{trace}(s_T^{-1} s_W) = \text{trace} \Lambda = \lambda_1 + \dots + \lambda_q, \quad (12)$$

where s_W represents the within-class matrix. This expression is maximized by sorting the eigenvalues $\{\lambda_1, \lambda_2, \dots\}$ in Λ in descending order and selecting the $K - 1$ largest ones and some conveniently normalized associated eigenvectors. Since the minimizer of (11) is not uniquely defined, it is usually normalized as $A^t S_T A = I_{K-1}$, being I_N the identity matrix of size N . It turns out that an equivalent solution can be obtained by solving the least squares problem $\min \frac{1}{2} \|Y^f - XW - \mathbf{1}_N B\|^2$, where W is a $d \times K$ matrix, B a $1 \times K$ vector, $\mathbf{1}_N$ is the all ones vector and for a pattern x^p in the p -th row of the data matrix X which is in class m , we have $Y_{pm}^f = \frac{N - N_m}{N \sqrt{N_m}}$ when x^p and as $Y_{pj}^f = -\frac{\sqrt{N_j}}{N}$ for $j \neq m$, with N_j the number of patterns in class j . Then, it can be shown that the optimal W^* is equivalent, up to a rotation, to a solution \widetilde{V} of (11) subject to the normalization $\widetilde{V}^t S_T \widetilde{V} = \Lambda$; see [4, 13] for more details. As a consequence, any classifier defined in terms of distances to class means will give the same results with the Fisher's projections using \widetilde{V} than with the least squares ones using W^* .

This can be extended to a DNN setting by solving

$$\min_{W, B, \widetilde{\mathcal{W}}} \frac{1}{2} \|Y^f - F(X, \mathcal{W})\|^2 = \frac{1}{2} \|Y^f - \Phi(X; \widetilde{\mathcal{W}})W - \mathbf{1}_N B\|^2. \quad (13)$$

As discussed in [4], this could be exploited to define a Fisher-like distance classifier on the network outputs $F(x, \mathcal{W})$; however, those classifiers are in general worse than those based on the categorical entropy or hinge losses. On the other hand, such a loss is likely to enforce the last hidden layer projections z to be concentrated around their class means while keeping these apart and, hence help entropy or hinge based classifiers to perform better.

In this line, the above suggests that instead of using just one of the previous losses, we can try to combine them, something that can be done by defining a

Table 1. Sample sizes, number of features and number of classes.

	Size	Size test	Features	Classes
a4a	4781	27780	123	2
a8a	22696	9865	123	2
australian	690	–	14	2
breast-cancer	569	–	30	2
diabetes	768	–	8	2
digits	1797	–	64	10
dna	2000	–	180	3
german	1000	–	24	2
letter	10500	5000	16	26
pendigits	7494	3498	16	10
protein	14895	6621	357	3
satimage	4435	–	36	6
segment	2310	–	19	7
usps	7291	–	256	10
w7a	24692	25057	300	2
w8a	49749	14951	300	2

DNN with two or even three output sets upon which one of these losses acts. In the most general setting, we may have outputs \hat{y}_{ce} , \hat{y}_{ch} and \hat{y}_f , one-hot encoded targets y and the y_f Fisher-like targets just defined for the loss (13), and we define the combined loss

$$\ell(y, y_f, \hat{y}_{ce}, \hat{y}_{ch}, \hat{y}_f) = \ell_{ce}(y, \hat{y}_{ce}) + \lambda \ell_{ch}(y, \hat{y}_{ch}) + \mu \ell_f(y_f, \hat{y}_f), \quad (14)$$

with $\{\lambda, \mu\}$ appropriately chosen hyperparameters. In our experiments next, we will consider the (ce, fisher), (hinge, fisher), (ce, hinge) and (ce, hinge, fisher) loss combinations; for simplicity we will just take $\lambda = \mu = 1$.

Table 2. Test accuracies of the models considered.

	ce	ce-fisher	hinge	hinge-fisher	ce-hinge	ce-hinge-fisher	max	min
a4a	84.38	84.36	84.39	84.40	84.43	84.45	84.45	84.36
a8a	85.10	85.51	84.96	84.93	85.20	85.09	85.51	84.93
australian	86.52	85.65	85.22	85.07	85.36	85.07	86.52	85.07
breast-cancer	97.72	98.07	96.84	97.19	96.49	96.31	98.07	96.31
diabetes	77.60	77.60	76.82	76.95	77.08	76.95	77.60	76.82
digits	98.22	98.44	98.39	98.27	98.50	98.22	98.50	98.22
dna	95.70	95.85	94.65	94.85	95.55	95.60	95.85	94.65
german	77.90	77.50	75.30	76.40	74.00	74.80	77.90	74.00
letter	95.38	95.28	95.42	95.66	94.98	95.42	95.66	94.98
pendigits	99.52	99.45	99.47	99.52	99.49	99.44	99.52	99.44
protein	69.78	69.76	66.49	66.71	67.62	67.07	69.78	66.49
satimage	91.09	90.55	91.54	91.25	91.75	91.25	91.75	90.55
segment	97.66	97.79	97.01	97.36	97.88	97.79	97.88	97.01
usps	97.79	97.93	97.82	97.53	97.65	97.68	97.93	97.53
w7a	98.79	98.84	98.83	98.81	98.83	98.84	98.84	98.79
w8a	98.97	98.99	98.84	98.88	99.00	99.00	99.00	98.84

3 Experimental Results

In this section we will describe the considered models, describe the datasets we will use, present our experimental methodology and results, and finish the section with a brief discussion.

3.1 Models Considered

We will consider six basic model configurations, involving different network outputs, losses and ways to make predictions, namely

- `ce`: the model uses softmax outputs and the categorical cross-entropy loss. Class labels are predicted as the index of the output with the largest a posteriori probability.
- `hinge`: the model uses linear outputs and the categorical hinge loss. Class labels are predicted as the index of the largest output.
- `ce_hinge`: the model uses two different outputs, one with softmax activations and the other with linear ones; the losses are the categorical cross-entropy and the categorical hinge, respectively. To get predictions, the softmax function is applied to the second output set, so that we can see the entire output vector as made of estimates of posterior probabilities (although this is not true for the second set, as no probability model is assumed for the hinge loss); class labels are predicted as the index of the output with the largest value.
- `ce_fisher`: the model uses two different output sets, one with softmax activations and the other with linear ones. The categorical cross-entropy is minimized on the first and the Fisher loss introduced in Sect. 2 on the second. Label predictions are computed on the first set, as the one with the largest a posteriori probability.
- `hinge_fisher`: the model uses two linear outputs, the first one to minimize the categorical hinge loss and the second one the Fisher loss. Class labels are predicted as the index of the largest output of the first set.
- `ce_hinge_fisher`: the model uses now three different outputs, a first one with softmax activations and the other two with linear outputs; the categorical cross-entropy, categorical hinge and Fisher losses are minimized, respectively, on each output set. Here the softmax function is also applied to the hinge loss linear outputs and class labels are predicted as the index of the first two outputs with the largest value.

3.2 Datasets

We will work with sixteen datasets, namely `a4a`, `a8a`, `australian`, `breast_cancer`, `diabetes`, `digits`, `dna`, `german`, `letter`, `pendigits`, `protein`, `satimage`, `segment`, `usps`, `w7a` and `w8a`; eight of them are multiclass and the rest binary. All are taken from the LIBSVM data repository, except when pointed out otherwise. Table 1 shows their train and test (when available) sample sizes, dimensions and the number of classes. We give more details about them below.

Table 3. Model rankings for each problem in ascending accuracies.

	ce	ce-fisher	hinge	hinge-fisher	ce-hinge	ce-hinge-fisher
a4a	5	6	4	3	2	1
a8a	3	1	5	6	2	4
australian	1	2	4	5	3	5
breast-cancer	2	1	4	3	5	6
diabetes	1	1	6	4	3	4
digits	5	2	3	4	1	5
dna	2	1	6	5	4	3
german	1	2	4	3	6	5
letter	4	5	2	1	6	2
pendigits	1	5	4	1	3	6
protein	1	2	6	5	3	4
satimage	5	6	2	3	1	3
segment	4	2	6	5	1	2
usps	3	1	2	6	5	4
w7a	6	1	3	5	3	2
w8a	4	3	6	5	1	1
ave	3	2.6	4.2	4	3.1	3.6

- **a4a** and **a8a**. Variations of the `adult` of predicting whether income exceeds \$50,000 per year based on census data.
- **australian**. The goal is to decide whether or not an application is credit-worthy.
- **breast_cancer**. The goal is here to predict whether a patient is to be diagnosed with cancer.
- **diabetes**. The objective here is to diagnose the presence of hepatitis on a sample of Pima Indian women.
- **digits**. We want to classify pixel rasters as one of the digits from 0 to 9; the subset is pre-loaded in the *scikit-learn* library.
- **dna**. The goal is to classify splice-junction gene sequences into three different classes.
- **german**. This is another problem where patterns are to be classified as either good or bad credits.
- **letter**. Pixel displays are to be identified as one of the 26 English capital letters.
- **pendigits**. Images of handwritten digits between 0 and 9 are to be classified.
- **satimage**. The goal is to classify the central pixel in a satellite image; we will work only with the 4,435 training subsample.
- **segment**. We want to classify satellite images into one of seven categories.
- **usps**. We want to classify image rasters as a digit between 0 and 9.
- **w7a** and **w8a**. Variants of a classification problem of web pages.

3.3 Experimental Methodology and Results

Recall that all model losses include a L_2 regularization term, which requires the selection of a hyperparameter α so we will proceed first to estimate the optimal

Table 4. Model rankings for each problem in ascending accuracies after putting together models with closer rankings.

	ce	ce-fisher	ce-hinge	ce-hinge-fisher	hinge	hinge-fisher
a4a	2.0	3.0	1.0	1.0	3.0	2.0
a8a	3.0	1.0	2.0	1.0	2.0	3.0
australian	1.0	2.0	3.0	2.0	1.0	2.0
breast-cancer	2.0	1.0	3.0	3.0	2.0	1.0
diabetes	1.0	1.0	3.0	1.0	3.0	1.0
digits	3.0	2.0	1.0	3.0	1.0	2.0
dna	2.0	1.0	3.0	1.0	3.0	2.0
german	1.0	2.0	3.0	3.0	2.0	1.0
letter	1.0	2.0	3.0	2.0	2.0	1.0
pendigits	1.0	3.0	2.0	3.0	2.0	1.0
protein	1.0	2.0	3.0	1.0	3.0	2.0
satimage	2.0	3.0	1.0	2.0	1.0	2.0
segment	3.0	2.0	1.0	1.0	3.0	2.0
usps	2.0	1.0	3.0	2.0	1.0	3.0
w7a	3.0	1.0	2.0	1.0	2.0	3.0
w8a	3.0	2.0	1.0	1.0	3.0	2.0
ave	1.9	1.8	2.2	1.8	2.1	1.9

Table 5. Accuracy spreads across all models as percentages of the difference between the maximum and minimum accuracies over the minimum one.

	a4a	a8a	austr	breast	diab	digits	dna	german
Spread	0.11	0.69	1.7	1.82	1.02	0.28	1.27	5.27
	letter	pendig	protein	satimage	segment	usps	w7a	w8a
Spread	0.72	0.08	4.95	1.32	0.89	0.41	0.05	0.16

α and then evaluate model performance. Eight datasets considered have train-test splits and we will find the optimal α by searching on a one-dimensional logarithmically equally-spaced grid using 5-fold cross validation (CV) on the training set. Then we will evaluate the performance of optimal α^* model by computing its accuracy on the test set. On the other datasets we will apply 5-fold nested cross validation (CV), defining first a 5-fold outer split and applying again 5-fold CV to estimate the optimal α_i^* over the i -th outer train split. Once this α_i^* is obtained, the associated model is trained over the i -th outer train fold and applied on the patterns remaining on the i -th test fold; these class predictions \hat{y} are then compared with the true target labels y to compute now the accuracy of the model under consideration.

The resulting accuracies are given in Table 2 while Table 3 shows for each problem the model ranking by decreasing accuracies; when two or more give the same accuracy, they receive the same rank. We remark that these rankings are given only for illustration purposes and they do not imply statistically significant differences. This table also shows the mean ranking of each model across all the datasets considered. As it can be seen, the model with the best mean ranking is `ce-fisher`, followed by `ce` and `ce-hinge`. The following one is `ce-hinge-fisher` while `hinge` and `hinge-fisher` perform similarly but behind all others.

These similar performances can also be seen in Table 4, where model rankings are shown after we group together `ce-fisher`, `ce` and `ce-hinge` on a first model group, and `ce-hinge-fisher`, `hinge` and `hinge-fisher` on another. Here `ce-fisher` still performs best on the first group, while in the second `ce-hinge-fisher` and `hinge-fisher` perform similarly and better than `hinge`. In any case, the test accuracies of all models shown in Table 2 are quite similar. This can also be seen in Table 5, which shows for each problem the difference between the highest accuracy (i.e., the best one) and the smallest one (i.e., the worst) as a percentage of the latter. As it can be seen, and except for the `german` and `protein` problems, in all other this percentage is below 2%, and even below 1% in nine problems.

Finally, in Table 6 we give the statistic values returned by the Wilcoxon signed rank test when applied to the columns of the test accuracies in Table 2, and their associated p -values. To obtain them, we have sorted the different losses in increasing order of their rank averages given in the last row of Table 3; this means that in the rows of Table 6 the first model is better ranked than the second one and, hence, expected to perform better. Recall that the test’s null hypothesis is that the two paired samples come from the same distribution.

As it can be seen, this null hypothesis can be rejected at the $p = 0.05$ level when comparing the `ce-fisher` loss with the `ce-hinge-fisher`, `hinge-fisher` and `hinge`, and at the $p = 0.1$ level when doing so with the `ce-hinge` loss; on the other hand, the p -value when comparing it with the `ce` is quite high. This suggests that the `ce-fisher` loss performs similarly to the `ce` loss, but better than the others. In the same vein, the `ce` loss appears to perform better than the `hinge` and `hinge-fisher` ones; all the other loss pairings give similar performances.

Table 6. Model rankings for each problem in ascending accuracies after putting together models with closer rankings.

	stat	p-val
ce-fisher vs ce	62.0	0.776
ce-fisher vs ce-hinge	32.0	0.065
ce-fisher vs ce-hinge-fisher	27.0	0.036
ce-fisher vs hinge-fisher	23.0	0.018
ce-fisher vs hinge	22.0	0.016
ce vs ce-hinge	43.0	0.211
ce vs ce-hinge-fisher	39.0	0.14
ce vs hinge-fisher	24.0	0.024
ce vs hinge	27.0	0.034
ce-hinge vs ce-hinge-fisher	43.0	0.205
ce-hinge vs hinge-fisher	44.0	0.231
ce-hinge vs hinge	46.0	0.266
ce-hinge-fisher vs hinge-fisher	55.0	0.603
ce-hinge-fisher vs hinge	64.0	0.856
hinge-fisher vs hinge	44.0	0.231

3.4 Discussion

The preceding results indicate that the companion losses proposed here can improve on the accuracies of models based on the single `ce` and `hinge` losses. More precisely, adding the Fisher loss results in larger accuracies than those achieved by using just the `ce` and `hinge` ones, although this does not extend to the `ce-hinge` combination (which basically ties with the single `ce` loss), or when combining the `ce-hinge` with the Fisher loss, worsens the `ce-hinge` performance.

It is also to be pointed out that the performance of the `hinge`-based models is worse than that of the `ce`-based ones. A possible reason for this is the relatively small number of units in the single hidden layer architecture of all models. In fact, it is well known that for SVMs to achieve good results, the dimension of the projected input space (upon which an SVM acts linearly) must be quite large. For instance, in the case of the commonly used Gaussian kernel, this dimension essentially coincides with the sample’s size. Here we are using the last hidden layer activations as a proxy of the projection space but its dimension is 100, much below sample size for all problems. In fact, in [5] deep networks with at least 1,000 units in the last layer were needed to match or improve the performance of a standard Gaussian SVM. Also, better performance should also be possible with more hidden layers, although this will also help `ce` models. Finally, it is also clear that adding the Fisher loss helps; in fact, it seeks a last hidden layer representation which concentrates each class samples near their centroids while keeping these apart. This should help any classifier acting on that layer while, on the other hand, it doesn’t compete directly with `ce` and `hinge`.

4 Conclusions and Futher Work

In this paper we have proposed how to combine different classification losses in a single DNN so that each one acts on specific network outputs. The underlying goal is that these competing but, at the same time, complementary objectives result in models with a performance better than that of those built on each individual loss. We give experimental results on sixteen classification problems combining the categorical cross-entropy and hinge losses as well as a least squares one inspired in Fisher’s Discriminant Analysis, and they show that, indeed, such combinations yield better accuracies. In fact, using the Fisher based loss as a companion of the cross-entropy or hinge ones improved on the performance of DNN models using individually those losses; the same happens with the combination of the entropy and hinge losses.

These results suggest that further study is warranted. Beside the obvious extension to more complex network architectures than the simple one here, the choice of the hyperparameters used to define the combined loss (14) should be explored. Moreover, the same strategy of adding companion losses to a base one can be applied to regression problems, where natural choices are the mean square, ϵ -insensitive or Huber losses. We are currently pursuing these and related venues.

References

1. Abadi, M., et al.: TensorFlow: a system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2016), pp. 265–283 (2016)
2. Chollet, F.: Keras (2015). <https://github.com/fchollet/keras>
3. Crammer, K., Singer, Y.: On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.* **2**, 265–292 (2001)
4. Díaz-Vico, D., Dorronsoro, J.R.: Deep least squares fisher discriminant analysis. *IEEE Trans. Neural Netw. Learn. Syst.* **31**(8), 2752–2763 (2020)
5. Díaz-Vico, D., Prada, J., Omari, A., Dorronsoro, J.R.: Deep support vector neural networks. *Integr. Comput. Aided Eng.* **27**(4), 389–402 (2020)
6. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010. JMLR Proceedings, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010, vol. 9, pp. 249–256 (2010)
7. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015 (2015)
8. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25, 26th Annual Conference on Neural Information Processing Systems 2012, Proceedings of a Meeting Held 3–6 December 2012, Lake Tahoe, Nevada, United States, pp. 1106–1114 (2012)
9. Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc. (2019)
10. Tang, M., Perazzi, F., Djelouah, A., Ben Ayed, I., Schroers, C., Boykov, Y.: On regularized losses for weakly-supervised CNN segmentation. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 507–522 (2018)
11. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.: Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.* **11**, 3371–3408 (2010)
12. Weston, J., Watkins, C.: Support vector machines for multi-class pattern recognition. In: ESANN 1999, Proceedings of the 7th European Symposium on Artificial Neural Networks, Bruges, Belgium, 21–23 April 1999, pp. 219–224 (1999)
13. Zhang, Z., Dai, G., Xu, C., Jordan, M.I.: Regularized discriminant analysis, ridge regression and beyond. *J. Mach. Learn. Res.* **11**, 2199–2228 (2010)
14. Zhao, H., Gallo, O., Frosio, I., Kautz, J.: Loss functions for image restoration with neural networks. *IEEE Trans. Comput. Imaging* **3**(1), 47–57 (2017)

Chapter 6

Conclusions and further work

6.1 Conclusions

The conclusions for each of the previous chapters are:

- **Deep Neural Networks Prediction in Wind and Solar Energy:** While Deep Neural Networks are undeniably powerful when properly tuned and designed, especially if the topology of the data is taken into account, they are very computationally demanding and finding the correct combination of hyperparameter values can be a daunting task. In the particular case of prediction in renewable energy, Deep Neural Networks can take great advantage from the grid-like structure of the data with the use of convolutional layers, and the results can be further improved by the use of ensemble techniques. However, the use of specialized hardware to train them such as GPUs is a need due to the very high cost of their training.
- **Deep Fisher Discriminant Analysis and Imbalanced Classification:** Fisher Linear Discriminant Analysis is limited by its linear nature and the cost of the eigenanalysis it requires, making it unfeasible for large datasets. Its kernel extension, Kernel Discriminant Analysis, has much more powerful modeling capabilities, but has an even higher computational cost, which is cubic on the sample's size. These difficulties are alleviated if a different approach based on Least Squares Regression is followed. With this setup, a partially equivalent solution can be obtained by using an appropriate target matrix, and the cost is linear on the sample's size. Moreover, when a properly designed Artificial Neural Network is used instead of a linear regressor, the results are comparable to those of the kernel model, while the cost remains linear on the size of the dataset. Also, this approach enables the use of special layers in the underlying neural network that can be tailored to the dataset, like convolutional or recurrent layers, and therefore an improvement on the results might be expected in those cases where the dataset allows it. Although Fisher-based classifiers might not yield better results than cross-entropy-based ones, the representations they yield can be used to feed more powerful classifiers that can take advantage of them. Finally, imbalanced classification problems have been tackled with a variety of tailored methods, but Artificial Neural Networks

can also yield state of the art results when properly configured. In particular, this means using an appropriate error function to guide the hyper-parameter search, rectified activation functions or modern regularization techniques.

- **Deep Support Vector Machines and Companion Losses:** Margin-based loss functions have demonstrated their usefulness in Support Vector Machines. Unfortunately, the computational cost of training or inference in kernel SVMs is too high for big size datasets. However, Artificial Neural Networks can also take advantage of margin-based loss functions thanks to the great flexibility of modern Deep Learning frameworks, that can even perform automatic differentiation of piecewise differentiable functions. Therefore, if properly configured and tuned, an Artificial Neural Network could yield comparable or even superior results to those of Linear and Kernel Support Vector Machines at a fraction of the cost, both in classification and regression problems. Furthermore, the flexibility of neural networks can again be of advantage when dealing with spatial or temporal structured data, combining the powerful capabilities of convolutional or recurrent layers with margin-based loss functions. At last, and again taking advantage of the modern Deep Learning frameworks, the possibility of combining several loss functions on the same neural network can also be considered. If the losses combined complement each other, the performance of the model can improve significantly. In particular, the least squares loss inspired in Fisher's Discriminant Analysis discussed before can improve the results of classifiers when combined with other losses such as cross-entropy or hinge.

6.2 Further work

The powerful modeling abilities of ANNs and their great flexibility are out of doubt, but there are still many open lines of research and unexplored applications.

Of particular interest are feature transformation techniques which allow the decomposition of a complex Machine Learning model into simpler sub-models by splitting it into a feature transformer, that builds a good latent representation of the data, and a predictor, that produces a numerical, categorical or ordinal value. Besides the benefits derived from this possibility, which include an increased interpretability of the model, higher degree of control over it, or easier debugging, feature transformations are also useful by themselves, with direct applications such as lossy data compression.

However, the most widely used feature transformation models, such as Principal Component Analysis, Fisher's Linear Discriminant Analysis or Canonical Correlation Analysis have linear nature, and their training algorithms are often based on eigenanalysis and associated to a very poor scalability with the size of the data. Well known non-linear extensions make use of kernels, which greatly improve the expressive power of the models, but at the cost of an even higher computational complexity, resulting in techniques only useful for relatively small datasets.

The results obtained for Fisher's Discriminant Analysis suggest the interest of extending them to the just mentioned transformation methods. The ultimate goal

would be, as it is the case with Fisher analysis, to achieve an alternative characterization of an equivalent method that can be built in the linear case through an appropriate least-squares loss. A possible path for this starts with the observation that the solutions of the above problems can be achieved through the formulation of a particular generalized eigenvalue problem involving concrete sample-derived matrices.

As it was the case of Fisher analysis, a first step would then be to derive a related least squares problem with a solution equivalent in some sense to the one obtained solving directly the generalized eigenvalue problem corresponding to each transformation method.

Such a least squares problem could be then cast in a Deep Learning framework and result in richer, non linear representations of the original data which can be used to yield better results than those achieved by the initial linear models.

The resulting model performance will have to be compared with that achieved by the kernel versions of the initial linear models, but it is clear that their computational costs would be much better than those of the kernel models; hence, they could be applied to sample sizes outside the reach of kernel based methods.

Moreover, such an approach would also result in new companion losses that could be coupled, as done in this thesis, with more standard ones yielding possibly stronger and more robust classification or regression models.

We are starting to address these questions.

Chapter 7

Conclusiones y trabajo futuro

7.1 Conclusiones

Las conclusiones para cada uno de los capítulos previos son:

- **Deep Neural Networks Prediction in Wind and Solar Energy:** Aunque las Redes Neuronales Profundas son innegablemente potentes cuando están diseñadas y entrenadas correctamente, especialmente si la topología de los datos es tenida en cuenta, son muy costosas computacionalmente y encontrar la combinación correcta de valores de meta-parámetros puede ser una tarea ardua. En el caso particular de la predicción en energías renovables, las Redes Neuronales Profundas pueden beneficiarse de la estructura en forma de rejilla de los datos con el uso de capas convolucionales, y los resultados pueden ser mejorados más aún mediante el uso de ensembles. Sin embargo, el uso de hardware especializado para entrenarlas, como las GPUs, es una necesidad debido al alto coste de su entrenamiento.
- **Deep Fisher Discriminant Analysis and Imbalanced Classification:** El Análisis Discriminante Lineal de Fisher está limitado por su naturaleza lineal y el coste del cálculo de autovectores y autovalores que requiere, haciéndolo impracticable para conjuntos de datos grandes. Su extensión kernel, Kernel Discriminant Analysis, tiene unas capacidades de modelización mucho mayores, pero con un coste computacional incluso mayor, que es cúbico en el tamaño de la muestra. Estas dificultades se pueden evitar si se sigue un enfoque diferente basado en "Least Squares Regression". Con este enfoque se puede obtener una solución parcialmente equivalente mediante el uso de la matriz de targets apropiada, con un coste lineal en el tamaño de la muestra. Además, si se utiliza una Red Neuronal Artificial en lugar de un regresor lineal, los resultados son comparables con los de modelos Kernel, pero con un coste que sigue siendo lineal respecto del tamaño del conjunto de datos. Esta técnica también permite el uso de capas especiales en la red neuronal subyacente que pueden ser ajustadas a medida de los datos, como capas convolucionales o recurrentes, y por tanto se puede esperar mejoras en los resultados en los casos en los que los datos lo permitan. Aunque los clasificadores de tipo Fisher no suelen dar resultados mejores que aquellos basados en la entropía cruzada, las representaciones

que producen pueden ser utilizadas por clasificadores más potentes que se benefician de ellas. Finalmente, los problemas de clasificación desequilibrada han sido abordados con una variedad de métodos específicos, pero las Redes Neuronales Artificiales también pueden obtener resultados comparables al estado del arte si son configuradas correctamente. En particular, esto implica usar una función de error apropiada para guiar la búsqueda de meta-parámetros, activaciones rectificadas o técnicas de regularización modernas.

- **Deep Support Vector Machines and Companion Losses:** Las funciones de pérdida con margen han demostrado su utilidad en las Máquinas de Vectores de Soporte. Desafortunadamente, el coste computacional de entrenar o predecir con Máquinas de Vectores de Soporte con Kernel es demasiado alto para conjuntos de datos grandes. Sin embargo, las Redes Neuronales Artificiales también pueden beneficiarse de las funciones de pérdida con margen gracias a la gran flexibilidad de las herramientas modernas para Deep Learning, que pueden incluso hacer diferenciación automática de funciones diferenciables a trozos. Por tanto, si se configura y ajusta adecuadamente, una Red Neuronal Artificial puede dar resultados comparables o incluso superiores a los de las Máquinas de Vectores de Soporte lineales o con kernel con una fracción de su coste, tanto en problemas de clasificación como de regresión. Es más, la flexibilidad de las redes neuronales puede ser de nuevo una ventaja cuando se trata con datos de estructura espacial o temporal, combinando las capacidades de las capas convolucionales o recurrentes con funciones de pérdida con margen. Por último, y de nuevo aprovechando las herramientas modernas para Deep Learning, también se ha considerado la posibilidad de combinar varias funciones de pérdida simultáneas en la misma red neuronal. Si las pérdidas combinadas se complementan, el rendimiento del modelo puede mejorar significativamente. En particular, la pérdida de mínimos cuadrados inspirada en el Análisis Discriminante Lineal presentada previamente puede mejorar los resultados de los clasificadores cuando se combina con otras pérdidas como la entropía cruzada o hinge.

7.2 Trabajo futuro

La gran capacidad para modelizar de las Redes Neuronales Artificiales y su gran flexibilidad están fuera de duda, pero aún hay muchas líneas de investigación abiertas sobre ellas y aplicaciones no estudiadas.

Son de especial interés las técnicas de transformación de variables que permiten la descomposición de un modelo de Aprendizaje Automático complejo en sub-modelos más simples, como un transformador de variables que genere una buena representación latente de los datos, y un predictor que arroje los valores finales numéricos, categóricos u ordinales. Más allá de los beneficios derivados de esta posibilidad, que incluyen una mayor interpretabilidad del modelo, mayor control sobre él, o un mantenimiento más fácil, las transformaciones de variables son útiles en sí mismas, con aplicaciones directas como la compresión de datos con pérdida.

Sin embargo, los modelos de transformación de variables más extendidos, como el

Análisis de Componentes Principales, el Análisis Discriminante de Fisher o el Canonical Correlation Analysis, tienen naturaleza lineal, y sus algoritmos de entrenamiento están a menudo basados en un análisis de autovalores y autovectores que se asocian a una muy mala escalabilidad con el tamaño de los datos. Hay extensiones no lineales populares basadas en kernels, que mejoran notablemente la capacidad expresiva de los modelos, pero con un coste computacional aún mayor, resultando en técnicas sólo útiles cuando se trabaja con conjuntos de datos relativamente pequeños.

Los resultados obtenidos para el Análisis Discriminante de Fisher sugieren que puede ser interesante una extensión para los métodos de transformación mencionados. El objetivo final sería, como en el caso del análisis de Fisher, conseguir una formulación alternativa de un método equivalente en el caso lineal mediante la función de pérdida de mínimos cuadrados. Una posible vía para conseguirlo parte de la observación de que las soluciones a los problemas mencionados arriba se pueden obtener con una formulación particular del problema generalizado de autovalores con unas matrices concretas obtenidas a partir de los datos de entrenamiento.

Como en el caso del análisis de Fisher, un primer paso sería obtener del problema de mínimos cuadrados relacionado una solución parcialmente equivalente a la obtenida a través del problema de autovalores particular de cada método de transformación.

Este problema de mínimos cuadrados podría entonces resolverse mediante herramientas modernas de Deep Learning y producir unas representaciones no lineales más ricas de los datos originales, que por tanto producirían mejores resultados que aquellas dadas por los modelos lineales iniciales.

El rendimiento resultante deberá ser comparado con el obtenido con las versiones kernel de los métodos lineales iniciales, pero en cualquier caso sus costes computacionales serán mucho mejores que los de los métodos kernel y, por tanto, se podrán aplicar a muestras de datos de tamaño inasumible para estos.

Además, este enfoque también puede dar lugar a nuevas "companion losses" que, combinadas con funciones de pérdida estándar como se ha hecho en esta tesis, produzcan modelos de clasificación y regresión más robustos y potentes.

Estamos empezando a trabajar en estas líneas de investigación.

Bibliography

- [1] Bruce G. Buchanan and Edward H. Shortliffe. *Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project (The Addison-Wesley Series in Artificial Intelligence)*. Addison-Wesley Longman Publishing Co., Inc., USA, 1984.
- [2] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA, 2012.
- [3] Ronald Arkin. Moving up the food chain: Motivation and emotion in behavior-based robots. *Who Needs Emotions: The Brain Meets the Robot*, 01 2005.
- [4] Xiaojin Zhu. Semi-supervised learning literature survey, 2007.
- [5] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996.
- [6] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.
- [7] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA, 2014.
- [8] Alex J. Smola, Thilo T. Frieund, and Bernhard Schölkopf. Semiparametric support vector and linear programming machines. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II*, page 585–591, Cambridge, MA, USA, 1999. MIT Press.
- [9] Cynthia Rudin and David Carlson. The secrets of machine learning: Ten things you wish you had known earlier to be more effective at data analysis. *CoRR*, abs/1906.01998, 2019.
- [10] Ian T. Jolliffe. A note on the use of principal components in regression. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 31(3):300–303, 1982.
- [11] Robert C. Moore and John DeNero. L1 and l2 regularization for multiclass hinge loss models. In *Symposium on Machine Learning in Speech and Natural Language Processing*, 2011.

- [12] Leon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, page 161–168, Red Hook, NY, USA, 2007. Curran Associates Inc.
- [13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.
- [14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [15] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.
- [16] Gavin C. Cawley and Nicola L.C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *J. Mach. Learn. Res.*, 11:2079–2107, August 2010.
- [17] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, February 2012.
- [18] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, page 2951–2959, Red Hook, NY, USA, 2012. Curran Associates Inc.
- [19] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constr. Approx.*, pages 289–315, 2007.
- [20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [21] Jin Huang and Charles X. Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Trans. on Knowl. and Data Eng.*, 17(3):299–310, mar 2005.
- [22] David Díaz-Vico, Aníbal R. Figueiras-Vidal, and José R. Dorronsoro. Deep mlps for imbalanced classification. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2018.
- [23] Ana Azevedo and Manuel Filipe Santos. Kdd, semma and crisp-dm: a parallel overview. In Ajith Abraham, editor, *IADIS European Conf. Data Mining*, pages 182–185. IADIS, 2008.
- [24] Przemyslaw Biecek. Model development process. *CoRR*, abs/1907.04461, 2019.
- [25] European center for medium-range weather forecasts, <http://www.ecmwf.int/>.

- [26] David Díaz-Vico, Alberto Torres-Barrán, Adil Omari, and José R. Dorronsoro. Deep neural networks for wind and solar energy prediction. *Neural Processing Letters*, page 1–16, 04 2017.
- [27] David Díaz, Alberto Torres, and José R. Dorronsoro. Deep neural networks for wind energy prediction. In *Advances in Computational Intelligence - 13th International Work-Conference on Artificial Neural Networks, IWANN 2015, Palma de Mallorca, Spain, June 10-12, 2015. Proceedings, Part I*, pages 430–443, 2015.
- [28] David Díaz-Vico and José R. Dorronsoro. Deep least squares fisher discriminant analysis. *IEEE Transactions on Neural Networks and Learning Systems*, 31(8):2752–2763, 2020.
- [29] David Díaz-Vico, Adil Omari, Alberto Torres-Barrán, and José Ramón Dorronsoro. Deep fisher discriminant analysis. In Ignacio Rojas, Gonzalo Joya, and Andreu Catala, editors, *Advances in Computational Intelligence*, pages 501–512, Cham, 2017. Springer International Publishing.
- [30] David Díaz-Vico, Jesús Prada, Adil Omari, and José R. Dorronsoro. Deep support vector neural networks. *Integrated Computer-Aided Engineering*, pages 389–402, Jan 2020.
- [31] David Díaz-Vico, Jesús Prada, Adil Omari, and José R. Dorronsoro. Deep support vector classification and regression. In José Manuel Ferrández Vicente, José Ramón Álvarez-Sánchez, Félix de la Paz López, Javier Toledo Moreo, and Hojjat Adeli, editors, *From Bioinspired Systems and Biomedical Applications to Machine Learning*, pages 33–43, Cham, 2019. Springer International Publishing.
- [32] David Díaz-Vico, Ángela Fernández, and José R. Dorronsoro. Companion losses for deep neural networks. In *Hybrid Artificial Intelligent Systems*, pages 538–549, Cham, 07 2021. Springer International Publishing.
- [33] Zhihua Zhang, Guang Dai, Congfu Xu, and Michael I. Jordan. Regularized discriminant analysis, ridge regression and beyond. *Journal of Machine Learning Research*, 11:2199–2228, 2010.
- [34] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 249–256, may 2010.
- [35] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pages 408–415, 2008.
- [36] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING, 1998.

- [37] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [38] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K.R. Mullers. Fisher discriminant analysis with kernels. In *Neural Networks for Signal Processing IX: Proceedings of the 1999 IEEE Signal Processing Society Workshop (Cat. No.98TH8468)*, pages 41–48, 1999.
- [39] Sebastian Mika, Gunnar Rätsch, Jason Weston, Bernhard Schölkopf, Alexander J. Smola, and Klaus-Robert Müller. Invariant feature extraction and classification in kernel spaces. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 526–532, 1999.
- [40] Benyamin Ghogh, Fakhri Karray, and Mark Crowley. Fisher and kernel fisher discriminant analysis: Tutorial, 2019.
- [41] Jonathan H. Manton and Pierre-Olivier Amblard. A primer on reproducing kernel hilbert spaces. *Foundations and Trends in Signal Processing*, 8(1-2):1–126, 2015.
- [42] Warren S. McCulloch and Walter Pitts. Neurocomputing: Foundations of research. chapter A Logical Calculus of the Ideas Immanent in Nervous Activity, pages 15–27. MIT Press, Cambridge, MA, USA, 1988.
- [43] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.
- [44] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Representations by Back-Propagating Errors*, page 696–699. MIT Press, Cambridge, MA, USA, 1988.
- [45] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 5 2015.
- [46] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- [47] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pages 153–160, 2007.
- [48] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- [49] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, apr 2011.

- [50] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1139–1147. JMLR Workshop and Conference Proceedings, may 2013.
- [51] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.
- [52] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.
- [53] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- [54] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
- [55] Devansh Arpit, Yingbo Zhou, Bhargava Urala Kota, and Venu Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In *Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1168–1176, 2016.
- [56] Apache Software Foundation. Hadoop.
- [57] He Ma, Fei Mao, and Graham W. Taylor. Theano-mpi: a theano-based distributed training framework. *CoRR*, abs/1605.08325, 2016.
- [58] Google. Tensorflow, an open source software library for machine intelligence.
- [59] Ronan Collobert and Koray Kavukcuoglu. Torch7: A matlab-like environment for machine learning. In *In BigLearn, NIPS Workshop*, 2011.
- [60] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, jun 2010. Oral Presentation.
- [61] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.
- [62] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

- [63] Frank Seide and Amit Agarwal. CNTK: microsoft’s open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, page 2135, 2016.
- [64] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.
- [65] Ian J. Goodfellow, David Warde-Farley, Pascal Lamblin, Vincent Dumoulin, Mehdi Mirza, Razvan Pascanu, James Bergstra, Frédéric Bastien, and Yoshua Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.
- [66] F. Chollet. Keras: Deep learning library for theano and tensorflow, 2015.
- [67] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. cite arxiv:1912.01703Comment: 12 pages, 3 figures, NeurIPS 2019.
- [68] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.
- [69] N. Kruger, P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, A.J. Rodriguez-Sanchez, and L. Wiskott. Deep hierarchies in the primate visual cortex: What can we learn for computer vision? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1847–1871, Aug 2013.
- [70] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013.
- [71] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.
- [72] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.
- [73] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, dec 1989.
- [74] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

- [75] Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104, New York, 8 1960. Institute of Radio Engineers, Institute of Radio Engineers.
- [76] Lin Wu, Chunhua Shen, and Anton van den Hengel. Deep linear discriminant analysis on fisher networks: A hybrid architecture for person re-identification. *Pattern Recognition*, 65:238–250, 2017.
- [77] Vladyslav Sydorov, Mayu Sakurada, and Christoph H. Lampert. Deep fisher kernels – end to end learning of the fisher kernel gmm parameters. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1402–1409, Washington, DC, USA, 2014. IEEE Computer Society.
- [78] Jian Yang, A. F. Frangi, Jing-Yu Yang, David Zhang, and Zhong Jin. Kpca plus lda: a complete kernel fisher discriminant framework for feature extraction and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(2):230–244, Feb 2005.
- [79] David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurons in the cat’s striate cortex. *Journal of Physiology*, 148:574–591, 1959.
- [80] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.
- [81] Vina Ayumi, L.M. Rere, Mohamad Ivan Fanany, and Aniaty Arymurthy. Optimization of convolutional neural network using microcanonical annealing algorithm. 10 2016.
- [82] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. *CoRR*, abs/1502.04681, 2015.
- [83] James Martens. Deep learning via hessian-free optimization. In Johannes Fürnkranz and Thorsten Joachims, editors, *ICML*, pages 735–742. Omnipress, 2010.
- [84] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*, pages 257–269, 2010.
- [85] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010.
- [86] American Meteorological Society. 2013-2014 solar energy prediction contest, 2013.

- [87] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.
- [88] Zhihua Zhang, Guang Dai, and Michael Jordan. A flexible and efficient algorithm for regularized fisher discriminant analysis. pages 632–647, 09 2009.
- [89] David Diaz-Vico. scikit-kda. <https://github.com/daviddiazvico/scikit-kda>, 2017.
- [90] Scipy: Scientific computing tools for python.
- [91] Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, and Jürgen Schmidhuber. The Sacred Infrastructure for Computational Research. In Katy Huff, David Lippa, Dillon Niederhut, and M Pacer, editors, *Proceedings of the 16th Python in Science Conference*, pages 49 – 56, 2017.
- [92] Alberto Torres-Barrán, David Díaz-Vico, and José R. Dorronsoro. Sparse one hidden layer mlps. In *ESANN*, 02 2014.
- [93] David Diaz-Vico. scikit-datasets. <https://github.com/daviddiazvico/scikit-datasets>, 2017.
- [94] David Diaz-Vico. scikit-sacred. <https://github.com/daviddiazvico/scikit-sacred>, 2018.
- [95] David Diaz-Vico. scikit-keras. <https://github.com/daviddiazvico/scikit-keras>, 2017.
- [96] David Diaz-Vico. scikit-dda. <https://github.com/daviddiazvico/scikit-dda>, 2019.
- [97] Charles R. Harris, K. Jarrod Millman, St’efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern’andez del R’io, Mark Wiebe, Pearu Peterson, Pierre G’erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
- [98] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.
- [99] The pandas development team. pandas-dev/pandas: Pandas, February 2020.
- [100] Wes McKinney. Data Structures for Statistical Computing in Python. In St’efan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.
- [101] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[102] Michael Waskom and the seaborn development team. `mwaskom/seaborn`, September 2020.

Appendix A

Publications

This is a list of the articles published during the realization of this thesis, including their abstracts:

- Alberto Torres-Barrán, David Díaz-Vico, and José R. Dorronsoro. Sparse one hidden layer mlps. In *ESANN*, 02 2014 – We discuss how to build sparse one hidden layer MLP replacing the standard l2 weight decay penalty on all weights by an l1 penalty on the linear output weights. We will propose an iterative two step training procedure where the output weights are found using FISTA proximal optimization algorithm to solve a Lasso-like problem and the hidden weights are computed by unconstrained minimization. As we shall discuss, the procedure has a complexity equivalent to that of standard MLP training, yields MLPs with similar performance and, as a by product, automatically selects the number of hidden units.
- David Díaz, Alberto Torres, and José R. Dorronsoro. Deep neural networks for wind energy prediction. In *Advances in Computational Intelligence - 13th International Work-Conference on Artificial Neural Networks, IWANN 2015, Palma de Mallorca, Spain, June 10-12, 2015. Proceedings, Part I*, pages 430–443, 2015 – In this work we will apply some of the Deep Learning models that are currently obtaining state of the art results in several machine learning problems to the prediction of wind energy production. In particular, we will consider both deep, fully connected multilayer perceptrons with appropriate weight initialization, and also convolutional neural networks that can take advantage of the spatial and feature structure of the numerical weather prediction patterns. We will also explore the effects of regularization techniques such as dropout or weight decay and consider how to select the final predictive deep models after analyzing their training evolution.
- David Díaz-Vico, Alberto Torres-Barrán, Adil Omari, and José R. Dorronsoro. Deep neural networks for wind and solar energy prediction. *Neural Processing Letters*, page 1–16, 04 2017 – Deep Learning models are recently receiving a large attention because of their very powerful modeling abilities, particularly on inputs that have a intrinsic one- or two-dimensional structure that can be captured and exploited by convolutional layers. In this work we will apply Deep Neural Networks (DNNs) in two problems, wind energy and daily solar

radiation prediction, whose inputs, derived from Numerical Weather Prediction systems, have a clear spatial structure. As we shall see, the predictions of single deep models and, more so, of DNN ensembles can improve on those of Support Vector Regression, a Machine Learning method that can be considered the state of the art for regression.

- David Díaz-Vico, Adil Omari, Alberto Torres-Barrán, and José Ramón Dorronsoro. Deep fisher discriminant analysis. In Ignacio Rojas, Gonzalo Joya, and Andreu Catala, editors, *Advances in Computational Intelligence*, pages 501–512, Cham, 2017. Springer International Publishing – Fisher Discriminant Analysis’ linear nature and the usual eigen-analysis approach to its solution have limited the application of its underlying elegant idea. In this work we will take advantage of some recent partially equivalent formulations based on standard least squares regression to develop a simple Deep Neural Network (DNN) extension of Fisher’s analysis that greatly improves on its ability to cluster sample projections around their class means while keeping these apart. This is shown by the much better accuracies and g scores of class mean classifiers when applied to the features provided by simple DNN architectures than what can be achieved using Fisher’s linear ones.
- David Díaz-Vico, Aníbal R. Figueiras-Vidal, and José R. Dorronsoro. Deep mlps for imbalanced classification. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2018 – Classification over imbalanced datasets is a highly interesting topic given that many real-world classification problems present a concrete class with a much smaller number of patterns than the others. In this work we shall explore the use of large, fully connected and potentially deep MLPs in such problems. We will consider simple MLPs, with ReLU activations, softmax outputs and categorical cross-entropy loss, showing that, when properly regularized, these relatively straightforward MLP models yield state of the art results in terms of the areas under the ROC curve for both two-class problems (the usual focus in imbalanced classification) as well as for multi-class problems.
- David Díaz-Vico and José R. Dorronsoro. Deep least squares fisher discriminant analysis. *IEEE Transactions on Neural Networks and Learning Systems*, 31(8):2752–2763, 2020 – While being one of the first and most elegant tools for dimensionality reduction, Fisher linear discriminant analysis (FLDA) is not currently considered among the top methods for feature extraction or classification. In this paper, we will review two recent approaches to FLDA, namely, least squares Fisher discriminant analysis (LSFDA) and regularized kernel FDA (RKFDA) and propose deep FDA (DFDA), a straightforward nonlinear extension of LSFDA that takes advantage of the recent advances on deep neural networks. We will compare the performance of RKFDA and DFDA on a large number of two-class and multiclass problems, many of them involving class-imbalanced data sets and some having quite large sample sizes; we will use, for this, the areas under the receiver operating characteristics (ROCs) curve of the classifiers considered. As we shall see, the classification performance of

both methods is often very similar and particularly good on imbalanced problems, but building DFDA models is considerably much faster than doing so for RKFDA, particularly in problems with quite large sample sizes.

- David Díaz-Vico, Jesús Prada, Adil Omari, and José R. Dorronsoro. Deep support vector classification and regression. In José Manuel Ferrández Vicente, José Ramón Álvarez-Sánchez, Félix de la Paz López, Javier Toledo Moreo, and Hojjat Adeli, editors, *From Bioinspired Systems and Biomedical Applications to Machine Learning*, pages 33–43, Cham, 2019. Springer International Publishing – Support Vector Machines, SVM, are one of the most popular machine learning models for supervised problems and have proved to achieve great performance in a wide broad of predicting tasks. However, they can suffer from scalability issues when working with large sample sizes, a common situation in the big data era. On the other hand, Deep Neural Networks (DNNs) can handle large datasets with greater ease and in this paper we propose Deep SVM models that combine the highly non-linear feature processing of DNNs with SVM loss functions. As we will show, these models can achieve performances similar to those of standard SVM while having a greater sample scalability.
- David Díaz-Vico, Jesús Prada, Adil Omari, and José R. Dorronsoro. Deep support vector neural networks. *Integrated Computer-Aided Engineering*, pages 389–402, Jan 2020 – Kernel based Support Vector Machines, SVM, one of the most popular machine learning models, usually achieve top performances in two-class classification and regression problems. However, their training cost is at least quadratic on sample size, making them thus unsuitable for large sample problems. However, Deep Neural Networks (DNNs), with a cost linear on sample size, are able to solve big data problems relatively easily. In this work we propose to combine the advanced representations that DNNs can achieve in their last hidden layers with the hinge and ϵ insensitive losses that are used in two-class SVM classification and regression. We can thus have much better scalability while achieving performances comparable to those of SVMs. Moreover, we will also show that the resulting Deep SVM models are competitive with standard DNNs in two-class classification problems but have an edge in regression ones.
- David Díaz-Vico, Ángela Fernández, and José R. Dorronsoro. Companion losses for deep neural networks. In *Hybrid Artificial Intelligent Systems*, pages 538–549, Cham, 07 2021. Springer International Publishing – Modern Deep Neuronal Network backends allow a great flexibility to define network architectures. This allows for multiple outputs with their specific losses which can make them more suitable for particular goals. In this work we shall explore this possibility for classification networks which will combine the categorical cross-entropy loss, typical of softmax probabilistic outputs, the categorical hinge loss, which extends the hinge loss standard on SVMs, and a novel Fisher loss which seeks to concentrate class members near their centroids while keeping these apart.

Appendix B

Software

This thesis and the publications detailed before included experiments that required the development of the following open source libraries. All of them are compatible with the Scikit-learn library [20], which serves as the core tool of all the experiments:

- scikit-datasets [93] <https://github.com/daviddiazvico/scikit-datasets> – A collection of datasets from a variety of sources.
- scikit-sacred [94] <https://github.com/daviddiazvico/scikit-sacred> – Utility to perform Sacred [91] experiments.
- scikit-keras [95] <https://github.com/daviddiazvico/scikit-keras> – Scikit-learn compatible wrapper for Keras [66] artificial neural networks.
- scikit-kda [89] <https://github.com/daviddiazvico/scikit-kda> – Implementation of Kernel Discriminant Analysis.
- scikit-dda [96] <https://github.com/daviddiazvico/scikit-dda> – Implementation of Deep Discriminant Analysis.

Alongside Scikit-learn, several Python open source libraries have been used in this thesis:

- Scikit-learn [20] <https://scikit-learn.org/stable/index.html> – A Machine Learning library.
- Numpy [97] <https://numpy.org/> – The Python package for scientific computing.
- Scipy [90] <https://scipy.org/scipylib/> – A scientific and technical computing library.
- StatsModels [98] <https://www.statsmodels.org/stable/index.html> – A package for data exploration, estimation of statistical models, and performing statistical tests.
- Keras [66] <https://keras.io/> – A high-level Deep Learning framework.

- TensorFlow [58] <https://www.tensorflow.org/> – A low-level Machine Learning library used as the computing engine for Keras.
- Pandas [99, 100] <https://pandas.pydata.org/> – A library for data manipulation and analysis through tables and other data structures.
- Matplotlib [101] <https://matplotlib.org/> – A plotting library.
- Seaborn [102] <https://seaborn.pydata.org/> – A statistical data visualization library based on Matplotlib.
- Sacred [91] <https://github.com/IDSIA/sacred> – A library to help configure, organize, log and reproduce experiments.