Deep Neural Networks for Wind and Solar Energy Prediction

David Díaz–Vico[1] · Alberto Torres–Barrán[1] · Adil Omari[1] · José R. Dorronsoro[1]

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], Adadelta [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 impressing 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-paramater 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

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

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 $\mathrm{Var}(z_i) \simeq \mathrm{Var}(z_j)$ across the successive z_j layers and also that $\mathrm{Var}\left(\frac{\partial J}{\partial z_i}\right) \simeq \mathrm{Var}\left(\frac{\partial J}{\partial z_k}\right)$ 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 $\mathrm{Var}(W_i) = \frac{2}{M_i + M_{i+1}}$, 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

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

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

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

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 Adadelta), best results were obtained with Adadelta. 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-bssed 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].

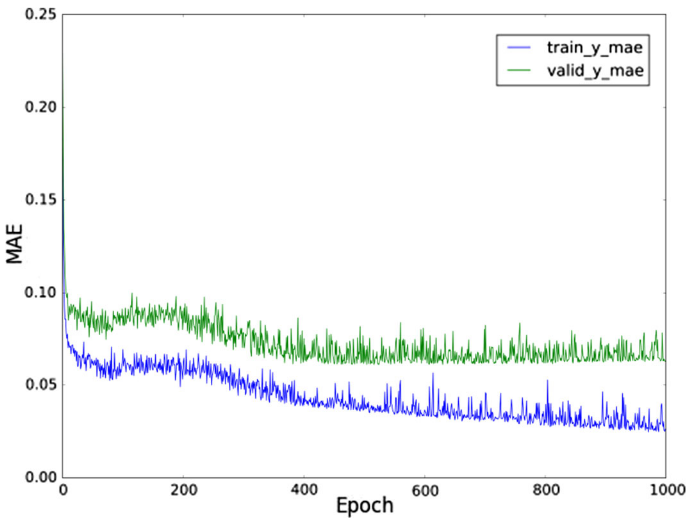

Fig. 1 Training and validation evolution of the optimal LeNet-5 network for Sotavento

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 Sotaveno 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 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

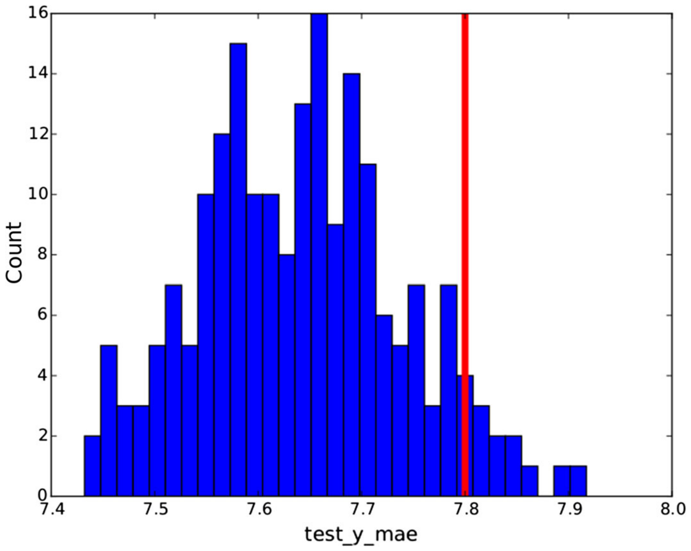

Fig. 2 Histogram of test errors over the individual 200 ensemble networks for the Sotavento problem

– 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:

- $apcp_s fc$, 3-h accumulated precipitation at the surface.
- $dlwrf_s fc$, downward long-wave radiative flux average at the surface.
- $dswrf_s fc$, downward short-wave radiative flux average at the surface.
- $pres_m sl$, air pressure at mean sea level.
- $pwat_e atm$, precipitable water over the entire depth of the atmosphere.
- $spfh_2 m$, specific humidity at 2 m above ground.
- $tcdc_e atm$, total cloud cover over the entire depth of the atmosphere.
- $tcolc_e atm$, total column-integrated condensate over the entire atmosphere.
- $tmax_2 m$, maximum temperature over the past 3 h at 2 m above the ground.
- $tmin_2 m$, mininmum temperature over the past 3 h at 2 m above the ground.
- $tmp_2 m$, current temperature at 2 m above the ground.
- $tmp_s fc$, temperature of the surface.
- $ulwrf_s fc$, upward long-wave radiation at the surface.
- $ulwrf_t atm$, upward long-wave radiation at the top of the atmosphere.
- $uswrf_s fc$, 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

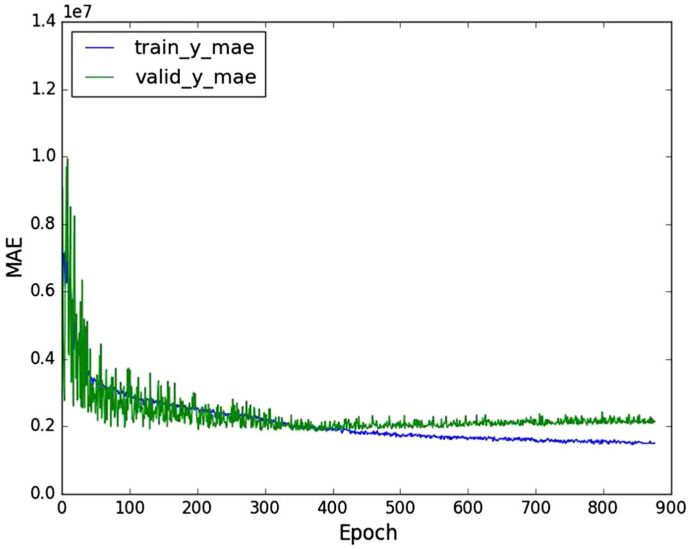

Fig. 3 Training and validation evolution of the optimal Keras–LeNet network for Solar

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

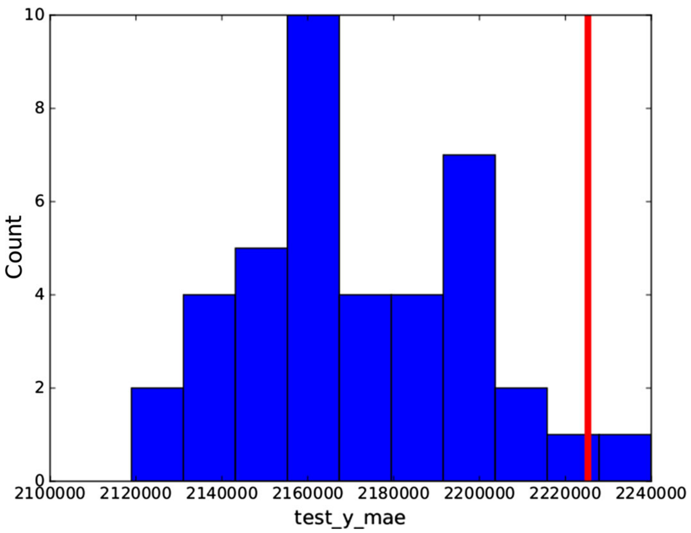

Fig. 4 Histogram of test errors over the individual 40 ensemble networks for the solar radiation problem

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 33nd 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
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/ioffe15.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