UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Degree as Compter Science Engineering

DEGREE WORK

Encoders for latent models with multiple, non-homomorphic realizations

Author: William Velez Martin
Advisor: Simone Santini
Co-advisor: Alberto Raimondi

junio 2021

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Codificadores para modelos latentes con múltiples realizaciones no homomórficas

Autor: William Velez Martin
Tutor: Simone Santini
Co-tutor: Alberto Raimondi

junio 2021

Francisco Tomás y Valiente, n^o 1
Madrid, 28049
Spain

William Velez Martin
Encoders for latent models with multiple, non-homomorphic realizations

William Velez Martin

PRINTED IN SPAIN

Agradecimientos

Para empezar, me gustaría destacar mi gratitud hacia mi tutor de este trabajo Simone Santini. Su orientación y apoyo han sido fundamentales para el progreso de este involucrado proyecto. Agradezco también la inestimable colaboración de su compañero italiano Alberto Raimondi, con quien empezó la idea fundamental de la que parto en mi TFG, y me indicó recursos teóricos excelentes con los que aprendí.

Especiales gracias a Pablo Castells por ofrecer recursos de un servidor con el cual trabajar con mis modelos, lo cual facilitó y aceleró mucho un proceso que podría haber tomado mucho más tiempo de manera innecesaria. También a Javier Sanz-Cruzado, quien fue tan amable de contactar conmigo y ofrecerme instrucciones de cómo acceder y utilizarlo.

Por último, quiero agredecer a mis amigos y familia por su incondicional apoyo, nunca dejaron de creer en mí y no podría haberlo hecho sin ellos.

Resumen

El procesamiento de lenguaje natural (NLP en inglés) es un campo de la inteligencia artificial (IA) puesto en práctica comúnmente en tareas que incluyen traducción, reconocimiento de voz, reconocimiento de escritura o incluso el etiquetado de funciones gramaticales (POS en inglés) de una palabra (verbos, sustantivos, etc.). El ”word embedding” es el proceso mediante el cual se representan palabras de cualquier manera matemática para su uso en tareas de NLP.

La idea detrás de este trabajo está relacionada: crear representaciones probabilísticas para las posibles funciones gramaticales que pueda desempeñar una palabra. Para ello, se usará un modelo especial llamado ”Conditioned Variational AutoEncoder” (CVAE), o autocodificador variacional condicionado. El CVAE se entrena para copiar su entrada creando un espacio latente desde el cual se muestra para generar la salida. En este caso la entrada es una palabra (y su contexto), y la variable a la que se condicionará su función gramatical. La función se puede extraer fácilmente como una etiqueta a través de bibliotecas como NLTK, pero la palabra debe tener algún tipo de representación para el modelo.

Un método popular es word2vec, el cual dado un gran cuerpo de texto genera vectores en un espacio para cada palabra. Una desventaja es que la información del orden de las palabras de contexto se pierde, y por tanto la sintaxis. Otra solución es el ”Transormer” BERT de Google, un modelo de lenguaje bidireccional entrenado para predecir palabras enmascaradas en una secuencia, o determinar si dadas dos secuencias, una es la continuación de otra. BERT también se puede usar para ”word embeddings”, mediante la extracción de los estados ocultos para cada palabra. Dada su naturaleza bidireccional, estos ”embeddings” sí que contienen información contextual a la derecha e izquierda de la palabra.

Aunque no de manera consistente, el CVAE es capaz de representar tres categorías gramaticales diferentes con tres distribuciones Gaussianas diferentes. Una serie de pesos se demuestra ser esencial para poder limitar cada categoría gramatical a un Gaussiano. Este trabajo puede ser expandido para representar significados de palabras en lugar de las funciones gramaticales, o incluso significados de palabras en diferentes idiomas.

Palabras clave

Deep learning, aprendizaje automático, procesamiento de lenguaje natural, transformadores, BERT, gaussiano

Abstract

Natural Language Processing (NLP) is a commonly used field of AI for tasks ranging from translation, speech recognition, handwriting recognition, or even part of speech (POS) tagging. Word embedding is the practice of representing words in a mathematical manner to perform NLP tasks. Typically this embedding is done through vectors.

The idea behind this thesis is related: to create probabilistic representations for a word's part of speech. For this, a special Conditioned Variational AutoEncoder (CVAE) will be used. The CVAE is trained to copy its input by creating a latent space from which the model will sample to generate its output, in this case a word (and its context). The variable it will be conditioned to is the POS tag. These can be easily obtained with libraries like NLTK, but the word that will be input to the model must have some sort of representation.

A popular approach is word2vec, which assigns a vectors representation to each word given a large text corpus. However, word2vec does not include contextual representation in its vectors. Another solution is Google's Transformer BERT, a bidirectional language model trained to predict a masked word in a sequence, and also determine whether two sequences are a continuation of each other. BERT can also be used to create word embeddings, by performing feature extraction of its hidden layers for any given word in a sequence. Because of its bidirectional nature, there embeddings contain ordered contextual information from the left and right.

Although not consistently, the CVAE is able to represent three different POS of a word with three different Gaussians. A series of weights that select one Gaussian when sampling from the latent space, prove to be essential to accomplish this task. This work could be expanded so that the representations are for word meanings, or even word meanings across multiple languages.

Keywords

Deep learning, machine learning, natural language processing, transformers, BERT, gaussian

Table of Contents

1 Introduction **1**

2 State of the art **3**

2.1 word2vec . . . 3

2.2 ELMo . . . 4

2.3 Transformer . . . 4

2.4 GPT-2 . . . 4

2.5 BERT . . . 5

2.6 Used technologies and resources . . . 5

3 Transformers and BERT **7**

3.1 Transformer's architecture . . . 7

3.2 BERT's architecture . . . 8

3.3 Input and output representations . . . 8

3.4 Pre-training tasks . . . 9

3.5 Fine-tuning . . . 10

3.6 Feature extraction . . . 11

4 Conditional Variational AutoEncoders **13**

4.1 AutoEncoder . . . 13

4.2 Variational AutoEncoder (VAE) . . . 14

4.3 Conditioned Variational AutoEncoders (CVAE) . . . 15

5 Clustering BERT word embeddings **17**

5.1 Data pre-processing and setup . . . 17

5.2 K-means clustering . . . 19

5.3 K-prototype and Gaussian Mixture Model clustering . . . 22

6 Probabilistic representations for parts of speech **23**

6.1 Predicting POS with a simple neural network . . . 23

6.2 Data pre-processing . . . 24

6.3 Conditioned Variational AutoEncoder . . . 24

6.4 Results . . . 25

7 Conclusions **29**

7.1 Future work . . . 30

Bibliography **32**

Lists

List of algorithms

List of codes

List of equations

4.1 Maximizing function for a VAE . . . 14
4.2 Log data likelihood for a VAE . . . 14
4.3 Minimizing term for a VAE . . . 14
4.4 Generator model for a CVAE . . . 15
4.5 Data likelihood for a CVAE . . . 15
4.6 Encoder model for a CVAE . . . 15
4.7 Gaussian selection function . . . 16
4.8 Decoder model for a CVAE . . . 16

List of figures

3.1 Transformer network architecture . . . 8
3.2 BERT's input representation . . . 9
3.3 BERT's pre-training and fine-tuning . . . 10
3.4 BERT's hidden layers . . . 11
3.5 Combining BERT's hidden layers . . . 11

4.1 AutoEncoder network architecture . . . 13

5.1 Silhouette scores of K-means for different K values . . . 19
5.2 K-means clusters and WordNet meanings for `right` . . . 20
5.3 K-means clusters showing WordNet meanings . . . 20
5.4 POS tags generated for the word `right` . . . 21
5.5 K-means clusters showing POS tags . . . 21
5.6 Clusters generated by a Gaussian Mixture Model and K-prototype model . . . 22

6.1 POS predicting neural network architecture . . . 23

6.2 CVAE implementation architecture 25
6.3 Gaussians generated by the encoder represented as both peaks and plotted μ values . 25
6.4 Gaussians generated by the encoder represented as both peaks and plotted μ values . 26
6.5 Gaussians generated by the encoder represented as both peaks and plotted μ values in Case 2 26
6.6 Gaussians generated by the encoder represented as both peaks and plotted μ values in Case 2 27
6.7 Gaussians generated by the encoder represented as both peaks and μ values in Case 2 27
6.8 CVAE implementation architecture without w_k weights 28
6.9 Gaussians generated by the encoder represented as both peaks and μ values in Case 3 28

List of tables

1 Introduction

Natural language processing (NLP) is increasingly becoming important part of Artificial Intelligence (AI) research. One common application today are virtual assistants such as Siri or Google Assistant. They must first use speech recognition in order to transform the spoken word to written language, after which they perform information extraction in order to correctly execute the user's command. Finally, they must generate a response that makes sense in the context request, through language generation.

The main topic of this thesis is equally language-related: to create a probabilistic representation of a word's ”part of speech” (POS), which is the grammatical function a word had in a sentence: noun, verb, adjective, etc. This will be accomplished with the help of a special type of AutoEncoder, a deep neural network designed to copy its input. Usually, the hidden states of this network have a smaller dimension than the input data. This compression forces the model to identify useful features to represent with its hidden state, which is just a vector of numbers. Variational AutoEncoders (VAE) on the other hand add a probabilistic spin by modeling the input data to a probability (or latent) space instead of vectors. The output is generated by sampling from this latent space.

The new idea is to condition the input to another variable, creating what is called a Conditioned Variational AutoEncoder (CVAE). It is the latent spaces that the CVAE generates what will represent a word's part of speech. POS tags can be easily extracted from a sentences with libraries like NLTK [1]. However, what can be used to transform a word into proper input for the CVAE network?

This task is what is called ”word embedding”. One type of word embedding could be frequency based, where a matrix is created for a sequence containing every unique word and its number of occurrences. However, embeddings are typically vector based like word2vec [1]. This algorithm makes use of neural networks to generate vector representations for every word in a given corpus. They are able to mathematically calculate synonyms or opposites of given words, and even answer to ”arithmetic” questions like ”King is to queen what prince is to X”, which can be re-written as ”X = king - queen + prince”. However, one thing that word2vec does not keep track of due to its architecture, is context word order. This may not be a problem with such unambiguous words like the ones just mentioned, but it becomes a clutch with other words that vastly change meaning based on context.

[1] https://www.nltk.org/

There is a different approach that can be taken to obtain word embeddings. Google's Transformer [2] is a type of network architecture that can process a sequence of words using exclusively attention mechanisms. These allow for the model to be aware of the context of each word it is processing, maintaining word order information. GPT-2 [3] is a Transformer based language model with over 1.5 billion parameters that is trained to predict the next word in a sequence. The one limitation GPT-2 has is that it only utilizes uni-directional context to the left of the word it is processing. On the other hand, Google's BERT [4] is also based on the Transformer, but unlike GPT-2 it is bi-directional in context. Since these language models can predict words, the hidden states of the network that give place to that prediction can be used as word embeddings, in a process called feature extraction.

BERT is a solid choice for this task because of its bi-directional nature, and the large corpus it was trained on (BookCorpus [5] and the English Wikipedia). This means that the hidden state for any word will have in it context information from both sides of the sequence, hopefully enough information to determine the POS of said word.

This thesis starts by exploring the use of BERT for feature extraction of word embeddings. It was originally considered to create probabilistic representations of word meanings instead of words parts of speech, but the experiments performed proved the latter to be an easier task to begin with. A Conditioned Variational AutoEncoder is presented and later implemented. It will end by discussing results and future work.

2

State of the art

In this thesis the main focus is on the field of Natural Language Processing (NLP). Many advances have been made in the last couple of years regarding language models and word embedding, for tasks such as text classification, sentiment analysis, information extraction or word embedding. The following are some of the most important recent advances.

2.1. word2vec

Developed by Google and released in 2018, word2vec [1] aims to create continuous vector representations for words given a large corpus. It consists of two different architectures:

Continuous bag-of-words model The CBOW predicts a target word based on a limited context to the left and right of it. However, the information about the order of these context words is lost.

Continuous skip-gram model The skip-gram model has the opposite task to the CBOW, where instead a word is used as input to predict the surrounding words.

Results show that it outperforms other neural network models (such as RNNs) on word similarity tasks at a much lower computational cost. It also learns different similarities between words, such as *Paris* being the capital of *France*, or *mice* being the plural of *mouse*. Given the vector nature of these representations, it is possible to perform basic arithmetic to find words matching a given relationship. For example, *Potter is to Gryffindor what Malfoy is to X* can be expressed as *X = Gryffindor - Potter + Malfoy*, where X should result in the vector corresponding to *Slytherin*.

It is clear word2vec is good at **semantic** tasks, but it suffers from a few weaknesses. The first of them is the lack of word order information, meaning its **syntax** capabilities are quite lacking. The second of these limitations is the fact that it becomes increasingly expensive to look in either direction for context words, where the paper uses a maximum of 20 total words of context. This means that long-term dependencies are not learned well.

2.2. ELMo

Contextualized word-embeddings are consolidated in the paper *Deep contextualized word representations* [6], where a new language model named ELMo (Embeddings from Language Model) is presented. It is based on an LSTM network and it is **bi-directional**, which means it is aware of the whole context of a word both to the left and right of it. This is achieved through two different models.The first is the *Forward language model* which starts at the beginning of the sequence, and the second is the *Backward language model* which starts at the end of the sequence.

ELMo's objective is to predict the next word in a given sequence of words. It follows a new approach however, where instead of only being a model like *word2vec*, it is a **pre-trained** model. Specifically, it is pre-trained on the large 1B Word Benchmark [7] for 10 epochs. This approach allows to leverage the information of such a large corpus, saving the user from the time and computational power that it would require to train it on their own. It is also possible to further fine-tune ELMo by adding new dense layers to the network, giving it a lot of flexibility.

2.3. Transformer

At the moment, the state of the art network architectures for Natural Language Processing were RNNs and LSTM networks. These models are inherently sequential, where one hidden state is determined by the previous one. This implies information can be lost in long sequences, requires a lot of memory and hurts parallelization. The transformer [2] is a new network architecture created by Google to approach language modeling and machine translation while solving these shortcomings. Figure 3.1 shows its internal components. Instead of processing a word sequence word by word like in LSTMs, the transformer works with the entire sequence in parallel. It exclusively uses attention mechanisms in conjunction with encoders and decoders, which enables the network to model long-term dependencies in the sequence regardless of their distance in the sequence. Experiment results showed that it performs better in machine translation than previous networks while taking less time to train.

2.4. GPT-2

Using the transformer as a base, OpenAI [8] opted for creating a network composed uniquely of its decoder layers. Their first model was GPT [9], trained on a very large amount of data (specifically BookCorpus [5]) in an unsupervised way, and then fine-tuned. However, the model that gained more popularity was their next one, GPT-2 [3]. It is essentially a scaled up version of GPT and comes in different sizes, the biggest of which has around 1.5 billion parameters. It was trained to predict the next word in a large dataset of Internet text of 40GB of size. It showed good performance and even impro-

vements on state-of-the-art on language modeling tasks such as machine translation, summarization, question answering and Winograd Schema Challange [10]. One notable characteristic of GPT-2 is that it is only a forward language model, meaning it is unidirectional. Even with its good performance, it is only conditioned on left context but not right context.

2.5. BERT

Google releases the BERT [4] (Bidirectional Encoder Representations from Transformers) model in 2018 with a similar but opposite approach to OpenAI's GPT-2. Instead of using only transformer decoders, it exclusively uses the encoders. The main feature of BERT is that it conditions text both to the left and right, making it a bidirectional language model. It is pre-trained with BookCorpus [5] and English Wikipedia. There are two tasks that it performs during this stage: masked LM prediction (MLM), and next sentence prediction (NSP). In MLM, a random percentage of tokens is masked and it is BERT's job to predict them. On the other hand, in NSP BERT is given two sentences and must predict whether or not "sentence Bïs the continuation of "sentence A". It does so using binary tags `IsNext` and `NotNext`. Additionally, BERT can be fine tuned by using its output as input to another deep neural network to complete specific tasks, such as text classification. In fact, BERT performed so well that it was implemented in Google Search, explained in a blog post [1]. Chapter 3 explains how BERT works in depth.

2.6. Used technologies and resources

Anaconda for Python 3.8

In order to have a good way of managing different Python virtual environments Anaconda was chosen. Its latest Python version available for macOS at March 2021 was 3.8.2. It is the version used for initial testing of BERT and text pre-processing tasks in a local M1 MacBook Pro laptop.

Google Colaboratory

Since the laptop's integrated graphics are not the best option for using BERT to make predictions or train a model, Google Colaboratory [2] was chosen. It is a cloud-based solution for running Python notebooks, and even offers free GPU power. The exact GPU is not always the same, but most of the time there was an Nvidia Tesla T4 assigned. The Python version is 3.7.10.

[1] https://blog.google/products/search/search-language-understanding-bert
[2] https://research.google.com/colaboratory/faq.html

Tensorflow 1.15.2

Given that BERT was tested with TensorFlow 1.11.0, the major TF version selected to operate with it was 1. Since Google Colaboratory only allows to change this version, TF 1.15.2 instead of the more desirable TF 1.11.0.

BookCorpus

BookCorpus [5] is a large dataset comprised by around 8,000 books written in English, and the main source of data for the experiments done in this thesis. It is used to obtain a large number of sentences to feed the BERT model and obtain internal representations of specific words. Since it is not hosted by its creators, a script to reproduce it was published to GitHub [3]. Fortunately, another user already used this code to create and host his own reproduction of BookCorpus, publishing the download link on a tweet [4]. It contains around 18,000 text files each corresponding to a book written in English, and weighs around 6 GB.

NLTK

In order to perform word tokenization, part-of-speech (POS) tagging, and word sense disambiguation (WSD) the Natural Language ToolKit (NLTK) was used. It provides a number of language related tasks and acess to over 50 corpora and lexical resources.

Keras

Keras is a high level library to code neural networks in Python, and can run on top of Tensorflow. In this thesis it is used to create the models in Chapter 6. It was chosen for its ease of use, and being so popular, it is simple to learn different aspects of it.

[3] https://github.com/BIGBALLON/cifar-10-cnn
[4] https://twitter.com/theshawwn/status/1301852133319294976

3

Transformers and BERT

The first step into developing the desired probabilistic representations of a word's different grammatical functions, is to obtain initial embeddings first. For this task BERT [4] was chosen because it can provide a contextual representation of a word in a sentence through feature extraction. Since BERT is an integral part of the development of this thesis, this chapter will focus on providing more insight into the architecture, pre-training and use cases that are relevant. Since it is based on Google's own transformer model, it makes sense to begin there.

3.1. Transformer's architecture

Google introduces the *transformer* in 2017 in their research paper *Attention Is All You Need* [2], which is the main source of the information for this section.

The Transformer applies the basic encoder-decoder structure seen in other research papers [11]. In the encoder an input sequence of symbol representations $(x_1, ..., x_n)$ is mapped to a sequence of continuous representations $z = (z_1, ..., z_n)$. The decoder will output one by one a symbol sequence $(y_1, ..., y_n)$. It is auto-regressive at each step because previously generated symbols are used as input to generate the next one. Both the encoder and decoder stack self-attention and point-wise fully connected layers. Since BERT only uses encoders, the description of the decoder will not be included. The architecture as a whole can be seen in Figure 3.1.

Encoder

Each of the $N = 6$ stacked layers is composed of two sub-layers. The first one implements a multi-head self-attention mechanism, while the second is a fully connected feed-forward layer. Each of these sub-layers normalizes its output, so it would look like $LayerNorm(x + Sublayer(x))$. In all cases, the dimensionality is $d_{\text{model}} = 512$.

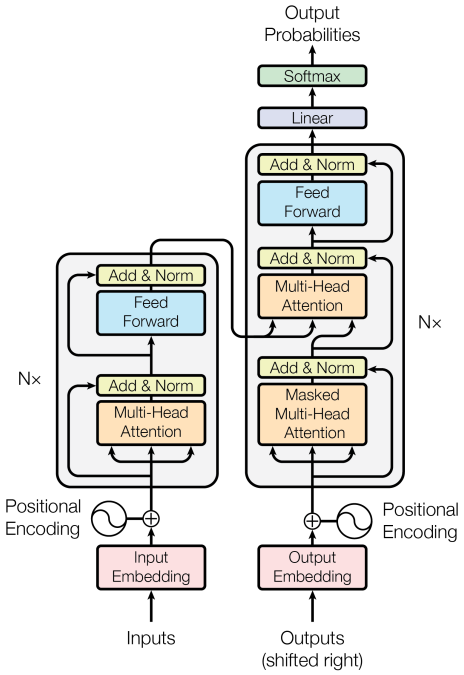


Figure 3.1: Transformer network architecture. To the left the encoder, and to the right the decoder. [2]

3.2. BERT's architecture

BERT's (Bidirectional Encoder Representations from Transformers) architecture is not complex once one has knowledge of the Transformer. As its own paper [4] explains, is a multi-layer bidirectional Transformer encoder. It is an encoder because it lacks the Transformer decoders altogether, with a hidden size of H. Multi-layer refers to the stack of these encoders (or Transformer blocks), whose size is defined by L. The number of self-attention heads is defined by A. The main model is $\mathbf{BERT_{base}}(L = 12, H = 768, A = 12, TotalParameters = 110M)$.

3.3. Input and output representations

Input

To begin with, BERT's input takes either 1 or 2 sentences as a token sequence. To be more specific, the either 1 or 2 sentences are tokenized with BERT's own tokenizer created with a WordPiece [12] model according to a fixed vocabulary size of 30,000 tokens. These tokens include the most common words and subwords in English. For example, the word *play* has its own token and it is tokenized as itself. However if the word *playing* were not in the vocabulary list, it would be divided into both *play* and *##ing*, where the latter is a common subword, indicated by the presence of *##*. The first token of

any sequence will always be `[CLS]`. It is a special token that corresponds to an aggregate sequence representation that can be used for classification tasks. Additionally, the token `[SEP]` is added at the end of each sentence in the sequence.

The final form of the input is the sum of these token embeddings, the positional embedding in the sentence, and the segment embedding (which is 1 or 0 depending if the token belongs to sentence 1 or 2). The input is zero-padded to accommodate the maximum sequence length of size H. A visual representation of the input can be seen in Figure 3.2.

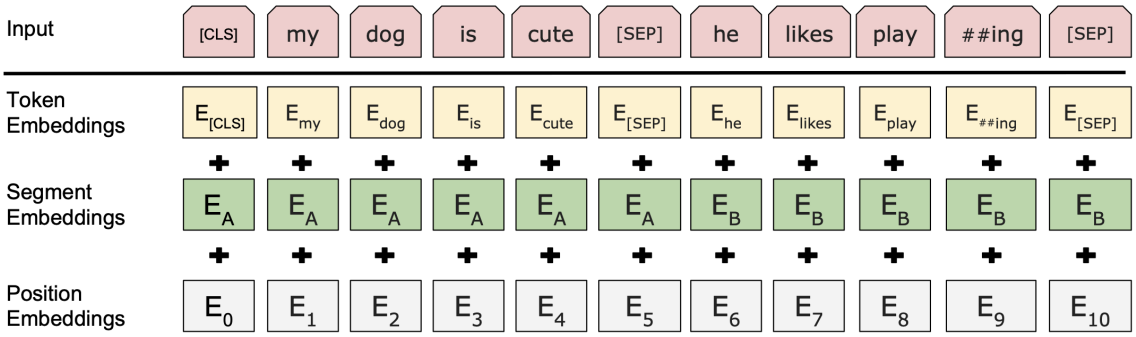


Figure 3.2: BERT's input representation [4]

Output

On the other, hand, the output is a vector of dimensions $seqlength \times H$. Each one of the components corresponding to a token can be used as a probability vector over the entire vocabulary in order to obtain that token. The first token [CLS] is meant to be used for classification since it corresponds to the representation of the entire sequence.

3.4. Pre-training tasks

Masked LM (MLM)

Standard conditional language models can either be trained left-to-right or right-to-left, but not both. BERT implements bidirectionality, which would allow each word to "see itself" and trivially predict the target word. To perform pre-training, 15 % of tokens are masked at random, and the model must predict these masked tokens. The rest of the input is not reconstructed.

Next sentence prediction (NSP)

The idea behind this task is given two sentences `A` and `B`, the model pre-trains for a binary label depending on whether or not `B` is a continuation of `A`. In 50 % of the training cases `A` will be a continuation of `B`, and the other 50 % it will not.

Pre-training data

Two main sources of data were used for pre-training: BookCorpus [5] which contains over 11,000 books and over 800 million words written in English in the form of text files, and the English Wikipedia with around 2.5 billion words. In Wikipedia's case, lists and tables were omitted with the purpose of extracting long contiguous sentences. Pre-training can be visualized in Figure 3.3.

3.5. Fine-tuning

Fine-tuning is a very simple process where the output of BERT can be fed onto another model as input designed to complete whatever task is desired. As explained earlier, for classification tasks it is the `[CLS]` that is forwarded to another model, since it is an aggregate representation of the one or two input sentences. Compared to pre-training, fine-tuning is not very computationally intense. The entire power of BERT can be easily leveraged for a multitude of tasks. Figure 3.3 shows this process.

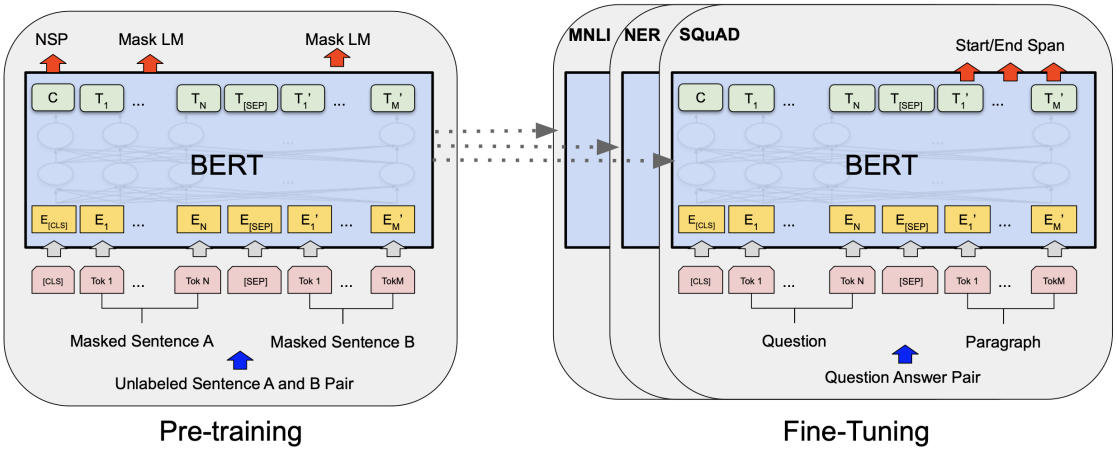


Figure 3.3: BERT's pre-training and fine-tuning [4]

3.6. Feature extraction

BERT can also be used for feature extraction. Since all hidden layers have the same dimension H, each of the components corresponds to a representation of a single token with the added advantage of having contextual bidirectional information. In order to obtain word embeddings, any of the hidden layer representations can be chosen. Additionally, the embedding can be obtained by combining representations of different layers in many ways, either by adding some of them, multiplying them or even concatenating them. An excellent visualization of this process was made by Jay Alammar in a post in his own blog [1], they can be seen in Figures 3.4 and 3.5.

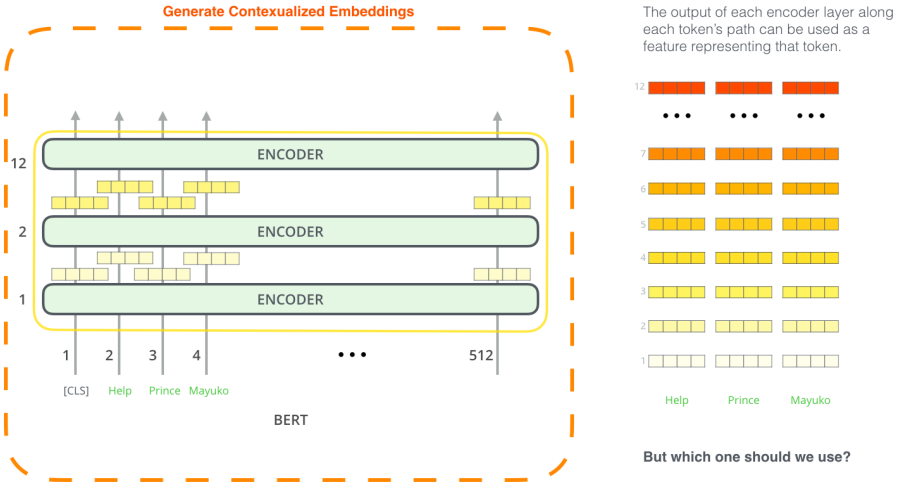


Figure 3.4: Intuition behind generating contextualized embeddings using BERT's hidden layers

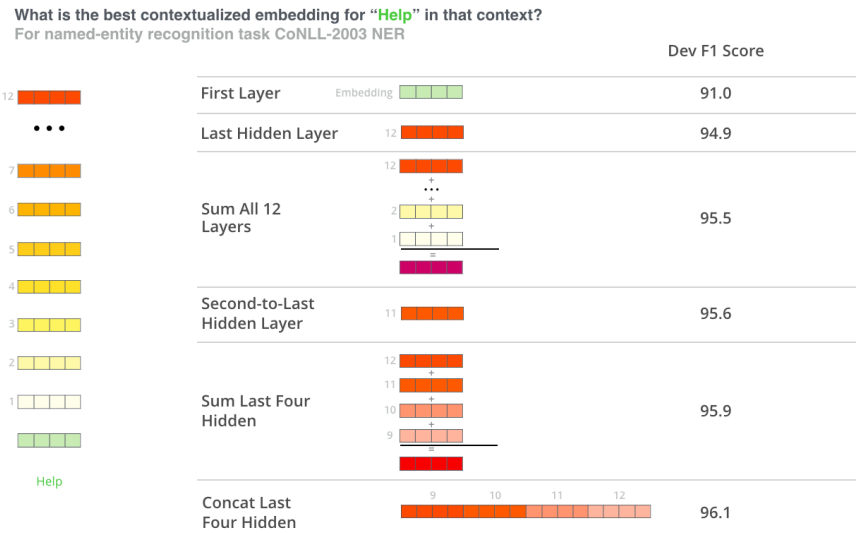


Figure 3.5: Different layer combinations and performance for contextualized word embeddings

[1] https://jalammar.github.io/illustrated-bert/

4

Conditional Variational AutoEncoders

Once BERT's [4] embeddings have been obtained, the next step is to create Gaussians that will be representative of different parts of speech (POS) of a word. For this purpose, a Conditioned Variational AutoEncoder was chosen (CVAE). This chapter will give more information on the technology and reasons behind choosing it, beginning with the original AutoEncoder, then the Variational AutoEncoders (VAE), and ending with the just mentioned CVAE.

4.1. AutoEncoder

Just as is explained in Ian Goodfellow, Yoshua Bengio, and Aaron Courville's book *Deep Learning* [14], a traditional AutoEncoder network is that which is trained in an unsupervised manner to copy its input. The **encoder** component maps an input x to a hidden representation $h = f(x)$, while the **decoder** will create a reconstruction of the input $r = g(h)$. A simple visual representation is provided by [14] that can be seen in Figure 4.1.

In order to obtain features from the input data, h can be constrained to have a smaller dimension than x. This type of AutoEncoder is called **undercomplete**. These can also be used as a compression method, whenever there can be information loss since an AutoEncoder cannot copy the input perfectly. The important element of this network is the hidden representation h, which in more clear terms is a vector in space and is usually called a **latent space** z.

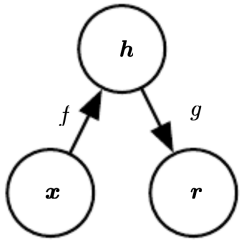


Figure 4.1: AutoEncoder network architecture [14]

4.2. Variational AutoEncoder (VAE)

The main sources of information for this chapter will be the original VAE paper [15] and a Stanford University online lecture [16]. Unlike the traditional AutoEncoder, a Variational AutoEncoder (VAE) adds a probabilistic element which will allow to generate data from the model. In a VAE, assume training data $\{x^{(i)}\}_{i=1}^{N}$ generated from an underlying latent space $\mathbf{z}$. This is done by sampling $p_{\theta^*}(x|z^{(i)})$ from the true prior $p_{\theta^*}(z)$, where θ are the parameters of the generative model that are going to be estimated. $p_{\theta^*}(z)$ is assumed to be a simple Gaussian, and $p_{\theta^*}(x|z^{(i)})$ can be represented as a **decoder** neural network. The objective is to learn parameters θ that maximize the data likelihood, which can be done through Equation 4.1. This however presents a problem because it is intractable to compute $p_\theta(x|z)$ for every $\mathbf{z}$.

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz \tag{4.1}$$

As a solution, an **encoder** network $q_\phi(z|x)$ is used to estimate $p_\theta(z|x)$, where ϕ are the parameters of the model. On the one hand, this **encoder** outputs distributions represented as the mean $\mu_{z|x}$ and (diagonal) covariance $\sum_{z|x}$, with $\mathbf{z}$ being sampled from $z|x \sim \mathcal{N}(\mu_{z|x}, \sum_{z|x})$. On the other hand, the **decoder** will output mean $\mu_{x|z}$ and covariance $\sum_{z|x}$, and then sample x|z from $x|z \sim \mathcal{N}(\mu_{x|z}, \sum_{x|z})$. The resulting log data likelihood can be expressed as in Equation 4.2, where the first two elements are the minimizing term or *evidence lower bound* as seen in Equation 4.3. More details of these equations can be seen in [15].

$$\log p_\theta(x^{(i)}) = \mathcal{L}(x^{(i)}, \theta, \phi) + D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)})) \tag{4.2}$$

$$\mathcal{L}(x^{(i)}, \theta, \phi) = \mathbf{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) \tag{4.3}$$

To make things more clear, the encoder will generate Gaussians (one per latent variable, composed by a mean and a standard deviation) that estimate $p_\theta(z|x)$, and the decoder samples from those distributions in order to reproduce the input as close as possible. Since the decoder is probabilistic, it can be used as a **generator** to create new data. However, this thesis is not concerned with the generator network, but rather the encoder or **inference** network, as the encoder is called sometimes. The objective is to make each of the Gaussians generated represent a different POS.

4.3. Conditioned Variational AutoEncoders (CVAE)

Motivation

The Conditioned Variational AutoEncoders (CVAE) is one of the fundamental aspects of this thesis as it is the method through which the Gaussian representations will be created. The source of the information and equations presented in this section is an unpublished work by Simone Santini [17], who proposes a model for a conditioned autoencoder that represents different word meanings with Gaussians, where an input word x is conditioned by its context words $\mathbf{Y} = [y_1, ..., y_n]$. In an effort to simplify the problem but still take advantage of the model, a decision was made to make this conditioned variable the one-hot vector of the POS tag. The idea is that a BERT feature vector already contains contextual information, so the tags should be able to facilitate each Gaussian to represent one of them. More details of this process is given in Chapter 5.

Theory

To begin with the theory, the basic principle of a CVAE is the same as a VAE, with the main difference being is that the generator model is conditioned to another variable, say $\mathbf{Y}$, as seen on Equation 4.4. The general idea is still similar to a VAE, given a dataset $\mathbf{X} = \{x^{(i)}\}_{i=1}^{N}$ it must find parameters θ that maximize the sum of each data sample likelihood, given by Equation 4.5. In the encoder or recognition model, a series of normalized weights determined by the conditional variable have the task of selecting one Gaussian (each being $q_i(z|x)$) that depends on the word x. The complete distribution then is a mixture of Gaussians, which can be seen in Equation 4.6.

$$p_\theta(x|\mathbf{Y}) = p_\theta(z|\mathbf{Y})p_\theta(x|z,\mathbf{Y})dz \tag{4.4}$$

$$\log p_\theta(x^{(i)}|\mathbf{Y}) = D_{KL}(q_\phi(z|x^{(i)},\mathbf{Y}); p_\theta(z|x^{(i)},\mathbf{Y})) + \mathcal{L}(\phi,\theta;x^{(i)},\mathbf{Y}) \tag{4.5}$$

$$q_\phi(z|x,\mathbf{Y}) = \sum_{k=1}^{K} w_k(\mathbf{Y})q_i(z|x) = \sum_{k=1}^{K} w_k(\mathbf{Y})\mathcal{N}(z|\mu_i(x),\sigma_i^2(x)\mathbf{I}) \tag{4.6}$$

Usually, sampling the Gaussians requires one independent variable $\epsilon \sim \mathcal{N}(0,\mathbf{I})$ such that $\mu + \sigma\epsilon$. This process is called reparameterization and allows back-propagating through sampling layers, more details can be seen in the original VAE paper [15]. In this particular implementation, the intention is to assign one Gaussian per POS tag, so a second independent variable $\nu \sim \mathcal{U}(0,1)$ is needed. When sampling q_ϕ, ν will select a Gaussian and ϵ will sample it.

The exact way this selection is made is through the series of functions shown in Equation 4.7, for $k = 1, 2, 3$ (since there will be 3 Gaussians to select from) where $w_0 = 0$. That way, the decoder or generator model $p_\theta(x|z)$ samples q as shown in Equation 4.8, where $f(x) = x$. With this theory, the experiments that will put it in practice can begin.

$$\theta_k(x) = \begin{cases} 1 & \sum_{h=0}^{k-1} w_h < x \leq \sum_{h=0}^{k} w_h \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

$$p_\theta(x|z) = \sum_k f_k(\theta_k(\nu)) g_k(\mu_k + \sigma_k \epsilon) \tag{4.8}$$

5 Clustering BERT word embeddings

5.1. Data pre-processing and setup

The first approach to this thesis was actually to use Gaussians to represent word meanings instead of POS tags, however this section will show why the former approach was changed for the latter. This chapter is meant to test the quality and properties of BERT's embeddings, by attempting to cluster them by word meaning. The expectation is that being contextual representations, the vectors generated for a single word should be able to cluster by its different meanings. As mentioned in Chapter 2, BookCorpus was the main source of data, so some pre-processing had to go behind it.

5.1.1. BookCorpus pre-processing

The objective is to cluster the different meanings of a word in the context of a whole sentence. Given that the original BookCorpus [5] contained over 74 million sentences, 30,000 seems like a good initial target number of sentences per word that will be clustered.

To accomplish this, a script developed for this purpose in Python 3 first shuffles a list containing all of the files in BookCorpus, and iterates through them. This is done in an effort to use a more varied selection of books. Without this shuffle, only the first couple of books would be analyzed in order and the rest would be wasted. This also allows to re-run the script multiple times with very little chance of obtaining the same sentences, which ensures access to a wide variety of data. It also allows to set a random seed for better control of the sentences obtained.

A limit is set on the number of sentences that can be extracted from each book, with the purpose of using as many files in the dataset as possible. In this case that limit is set to 10. Each file that contains a book is opened, and with the help of NLTK's punkt English tokenizer, is converted to a list of sentences.

Each is further divided into words with a regular expression from Python's `re` package. Now, all of those words are compared against a target word, and if its is found, the sentence it belongs to is added to a book sentence list. Once all 30,000 sentences are found they are written each in a line in a new text file named like `target_word.txt`. BERT can use this **word file** and its 30,000 lines as input.

5.1.2. Extracting BERT's word embeddings

A specific model size of BERT [4] has to be chosen. As a first contact, $\mathbf{BERT}_{\mathbf{base}}(L = 12, H = 768, A = 12)$ seems like an appropriate option. A script is given in BERT's code in GitHub that allows to perform feature extraction, by outputting a JSON file with the hidden states of a given input. The JSON file however has a very large size by default, since it is storing (as text) a 768 dimension vector per each token in an input sequence for 30,000 of these sequences, resulting in several gigabytes. This is not at all efficient, so the script was modified to only store the vector for target word common for all sentences, and store them as a numpy vector file `target_word.npy`. The new output **vector file** has a more manageable size of around 300 MB. The process of creating each **vector file** is done through Google Colab with a GPU instance.

5.1.3. Obtaining word meanings and POS tags

In order to validate the quality of the clusters that will be made, NLTK's library is used to get both POS tags and word meanings. On the one hand, POS tags are generated with the `nltk.pos_tag()` method. On the other hand, the source for the word meanings is WordNet [18–20], an English lexical database created in 1998. NLTK offers functionality to perform word sense disambiguation and return a WordNet meaning.

5.1.4. Word sense ambiguity

Word meanings pose a problem of ambiguity. For example, WordNet registers a total amount of 15 different meanings for the word `time`. Some of the meanings are different enough to be considered such, especially when they are different parts of speech (POS) for example:

- `Synset('time.n.01') an instance or single occasion for some event`
- `Synset('time.v.04') regulate or set the time of`

Other times however, two different WordNet meanings could be argued to be the same:

- `Synset('time.n.01') an instance or single occasion for some event`
- `Synset('time.n.04') a suitable moment`

For this reason, it is convenient to reduce the number of meanings to a more broad and sensible number. In the case of this experiment, the `target_word` is `right`, which WordNet says there are 36 different meanings. This seems like too high of a number to have any significance, so the target number of meanings to cluster for will be 4 [1]:

[1] https://www.dictionary.com/browse/right

- **Right as a noun:** a just claim or title, whether legal, prescriptive, or moral.
- **Right as an adjective:** in conformity with fact, reason, truth, or some standard or principle; correct.
- **Right as an adverb:** exactly; precisely.
- **Right as a verb:** to put in proper order, condition, or relationship.

5.2. K-means clustering

K-means clustering was selected as a starting point for its simplicity. Using around 30,000 vectors of 768 dimensions each, an initial probing was done to determine the best number of clusters with the use of the **silhouette score**. This metric is used to estimate the quality of clusters generated by K-means, and is given by the formula $S = \frac{(b-a)}{max(a,b)}$, where a is the mean intra-cluster distance, and b is the mean nearest-cluster distance. The score ranges from 1 to -1, with 1 being the best score possible, and 0 indicating overlapping clusters. The results of a number of different cluster sizes is shown in Figure 5.1.

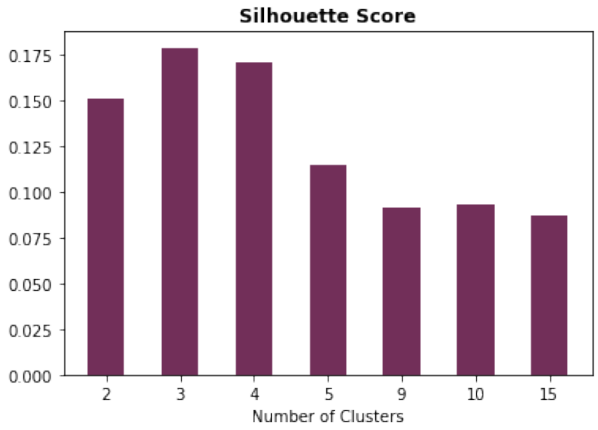
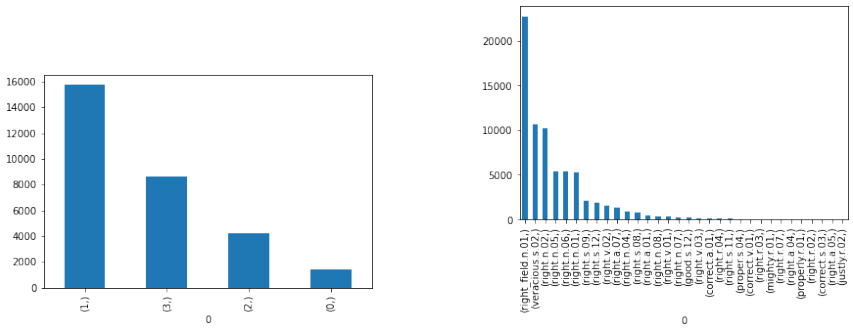


Figure 5.1: Silhouette scores of K-means for different K values

The initial estimation of 4 clusters was not too far off from the ideal case, however all scores in general are very low, with the highest not even reaching a score of $0{,}18$. Now, clustering with K-means where $K = 4$ creates 4 different groups shown in Figure 5.2(a). As a reference, Figure 5.2(b) shows the word meanings for `right` in the context of the original 30,000 sentences. There does not seem to be any correlation between the two. In fact, the WordNet meanings do not seem to reliable at all: most of the sentences were tagged with `('right_field.n.01') the piece of ground in the outfield on the catcher's right`, which would make sense if the source of the data were sports related, but this is not the case. Already, the task of Word Sense Disambiguation (WSD) proves to be a difficult one. To evaluate the quality of the meanings, below are a few examples of sentences tagged with `('right_field.n.01')`:

- She may look the part of the fashion plate right now but she was still a rancher's daughter at heart.
- All right, Astrid.
- She would, I suspected, be quite all right in the morning.
- I'm worried we're not going to get this all put together in time if El-Amin keeps demanding his speedy trial rights.
- You get your sorry ass back home right this minute!

None of these examples seem to fit the definition given by WordNet, so the entire quality of its meanings for this particular dataset is not very high. This is the main reason why the thesis topic moved away from word meanings. To make this point even more clear, Figure 5.3 shows the different word meanings contained in each cluster.



(a) K-means clusters

(b) Word meanings for the 30,000 sentences that contain `right`

Figure 5.2: To the left, a histogram showing the labels given to the `right` word vectors by K-means clustering. To the right, a histogram with the WordNet meanings obtained for the original sentences.

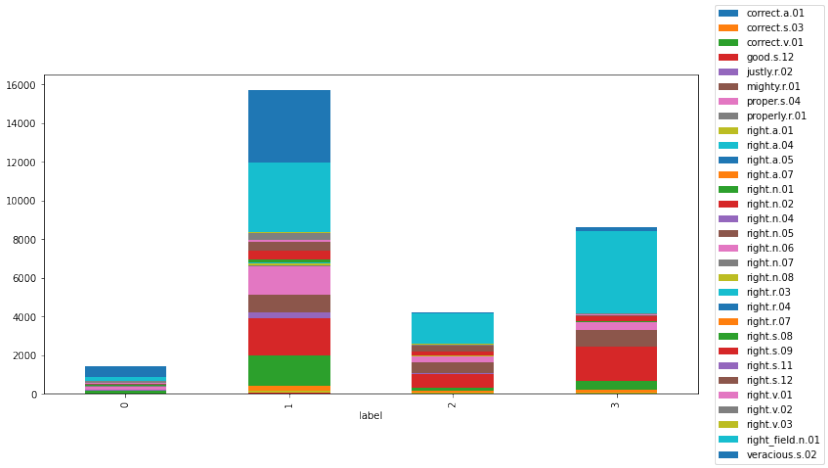
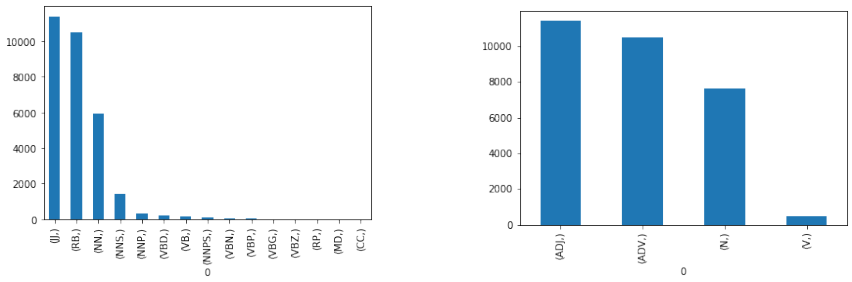


Figure 5.3: This figure shows the different WordNet meanings contained within the clusters created by K-means. It does not seem the clusters are very meaningful with respect to this metric.

This brings the attention to the POS tags generated by NLTK, seen in Figure 5.4(a), where `RB` stands for *adverb*, `JJ` stands for *adjective*, the ones starting with `N` are for nouns (split into plural, proper, etc.) and finally the ones starting with `V` stand for *verbs*. There are other tags present but they will be considered outliers. In order to further simplify these tags, nouns and verbs of different kinds are respectively grouped together into a single tag, shown in Figure 5.4(b). This distribution definitely seems to fit the data much better than WordNet's meanings. Below are some examples of sentences where `right` was tagged as a noun, and adjective:

- **Noun:** "We have no right to disclose information about our patients," the receptionist looked at the visitor intently.
- **Noun:** He was now at the stage in his recovery where he felt he didn't have a right to be there, in therapy, anymore.
- **Adjective:** Silver also took a more comfortable position, curling up on his chest, and took Kevin's hand with her left hand, still holding the gun with the right one.
- **Adjective:** His right hand had more freedom, its fingers lying on a small keyset with numbers from 0 to 9 and enter and cancel buttons.

Knowing that POS tags are more reliable, Figure 5.5 shows the POS distribution of the previously generated K-means clusters. While still not very meaningful, the task of dividing BERT's embeddings becomes easier.



(a) NLTK POS tags

(b) Simplified POS tags

Figure 5.4: These figures show the POS tags generated for the 30,000 sentences with the word `right`, both the original and simplified

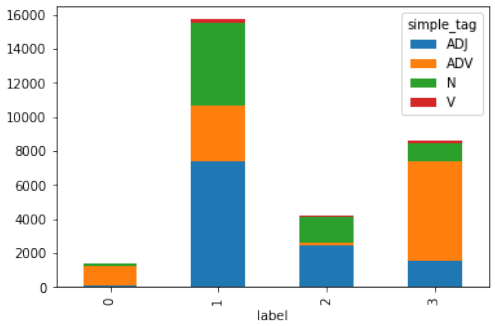


Figure 5.5: This figure shows the different POS tags contained within the clusters created by K-means.

5.3. K-prototype and Gaussian Mixture Model clustering

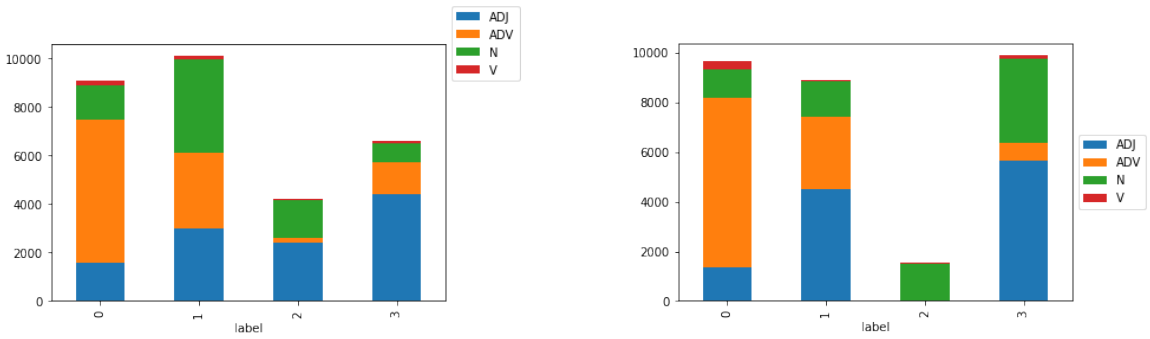
A few more clustering methods were utilized with BERT's word embeddings with the purpose of expanding on Section 5.2. This section will be kept short since results are largely the same as with K-means.

Gaussian Mixture Model (GMM)

The only parameter that must be specified in the Gaussian Mixture Model is the number of components, which was already accorded to be 4. Figure 5.6(a) shows the clusters created by the GMM along with the POS tags distribution between them. This method does not seem to be very revealing, since none of the clusters seem to depend on any POS information, and all of them include all different tags in varying proportions.

K-prototype

K-prototype allows to cluster a dataset containing both numerical and categorical values. For this particular method, the POS tags were concatenated to the `right` word vectors. Figure 5.6(b) shows the clusters it has created with its POS tag distribution. While not perfect, the clusters are of better quality: number 0 contains mostly adverbs, number 2 almost only includes nouns, and number 3 mostly includes a significant proportion of adjectives. This data seems to indicate that the POS tags do in fact offer useful information for BERT's embeddings, which will be used in the next chapter.



(a) GMM clusters

(b) K-prototype clusters

Figure 5.6: Clusters generated by a Gaussian Mixture Model and K-prototype model, along with their POS tags distribution.

6

Probabilistic representations for parts of speech

This chapter finally discusses the main topic of this thesis, that of creating probabilistic representations of POS tags for a word in different contexts. Details of the theory behind this can be found in Section 4.3 of this document. Here, the focus will be on the implementation and results obtained.

6.1. Predicting POS with a simple neural network

As the results in Chapter 5 suggest, parts of speech tags seem to complement BERT's hidden representations fairly well. In an effort to further prove this and justify its use later in section 6.3, a small feed-forward neural network was created to classify the `right` word vectors according to its POS tag. The architecture of said network can be seen in Figure 6.1. The optimizer is Adam with its default values, and the loss function is categorical cross entropy. The resulting model after being trained for 5 epochs can predict on a test set (30 % of the original data) with an accuracy of **83 %**. This serves as proof of concept that there is enough contextual data in BERT's hidden representations to predict a particular word's part of speech.

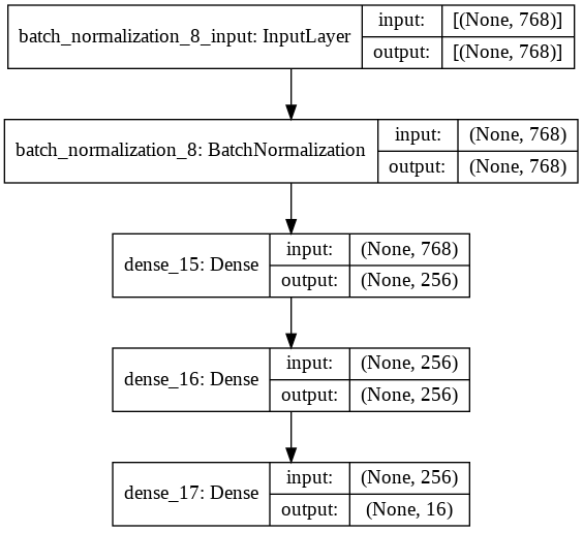


Figure 6.1: POS predicting neural network architecture

6.2. Data pre-processing

In order to simplify the problem in an effort to obtain results that could later be expanded upon, a new list of sentences containing the word `right` was generated from following the same process specified in Section 5.1.1. This time a file with 99,847 sentences was created, and its corresponding BERT word embeddings of dimension 768 were extracted and saved into a file with a total weight of 585 MB.

As seen in the simplified POS tags in Figure 5.4(b), the verbs are very small in number so the decision was made to remove them all together. They could hurt training performance as it would be noise to the other much more common tags, and there probably would not be enough cases for a Gaussian to fit it. This reduces the number of unique tags from 4 to 3, which means one less Gaussian in the model and in principe, make it easier for the Gaussians to fit a different tag each. After removing BERT's vectors corresponding to verbs, the total number is reduced to 98,161. Finally, these vectors were further reduced to 98,000 to facilitate compatibility with different batch sizes after splitting the data into training and validation sets with a 70/30 split.

6.3. Conditioned Variational AutoEncoder

As per the specifications in Section 4.3, a model was created with the use of Keras in Google Colab. The input for the **encoder** is a concatenation of a BERT vector and a one-hot encoded vector for the POS tag. One of its layer outputs 3 values for the mean of the Gaussian μ_k, another one outputs 3 standard deviations σ_k, and one last layer outputs the weights w_k that will select the Gaussian.

The **decoder** samples from these Gaussians in the following way: first a random variable $\nu \sim \mathcal{U}(0, 1)$ is passed to the function in Equation 4.7 (for which a custom TensorFlow function was created), which returns a vector of zeroes and one value of 1. Secondly, all $k = 3$ Gaussians are sampled in a vector with the random variable $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ where $\mu_k + \sigma_k \epsilon$. Lastly, both of these vectors are multiplied, ensuring only one of the Gaussians will have a value. The input for the decoder is the concatenation of this last vector containing one Gaussian, and once again the one-hot encoded vector for the POS tag. A visualization of the model can be seen in Figure 6.2

Training is done through a GPU instance in Google Colab, for 50 epochs and early stopping set with a patience value of 5 epochs.

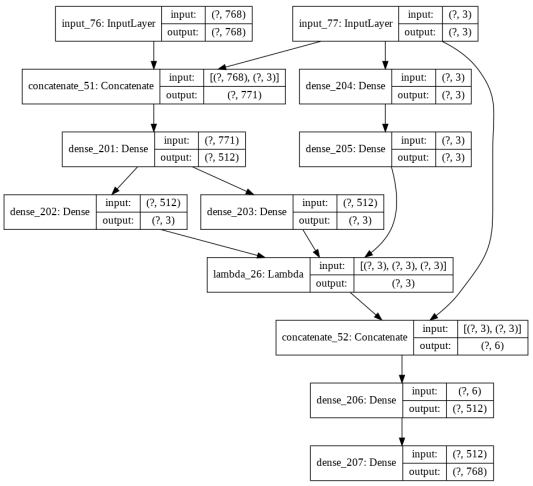
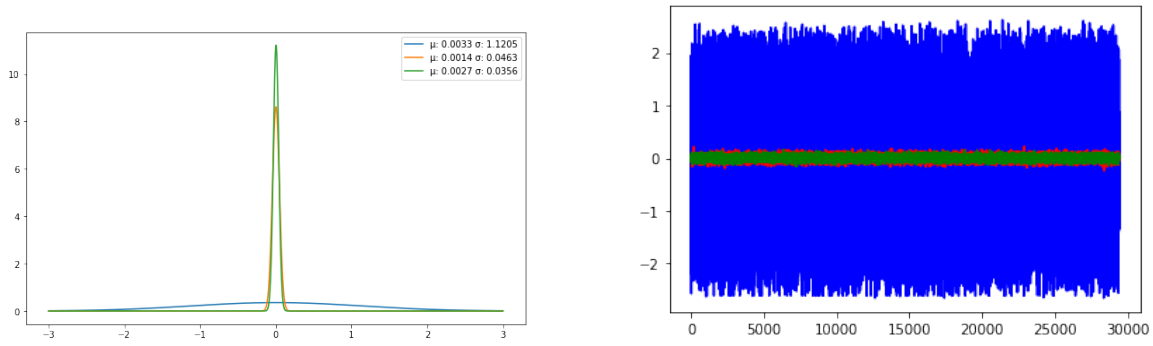


Figure 6.2: CVAE implementation architecture

6.4. Results

6.4.1. Case 1

By training the model various times for 50 epochs and setting Adam's learning rate to **0.001**, results show that only **one or two** of the Gaussians to converge to a peak. This indicates that only those are representing the POS tags for the input vectors. The way the Gaussians are extracted is by feeding input test data to the encoder and extracting the outputted μ values, which are then plotted according to its mean and standard deviation values. Figure 6.3(a) shows an example where two Gaussians converge to a peak, however they seem to very very close to each other approaching a mean of 0. Another visualization can be seen by simply plotting the μ points, seen in Figure 6.3(b), where it is clear that the *green* Gaussian is contained within the *red* one, and both of them contained within the *blue* Gaussian.

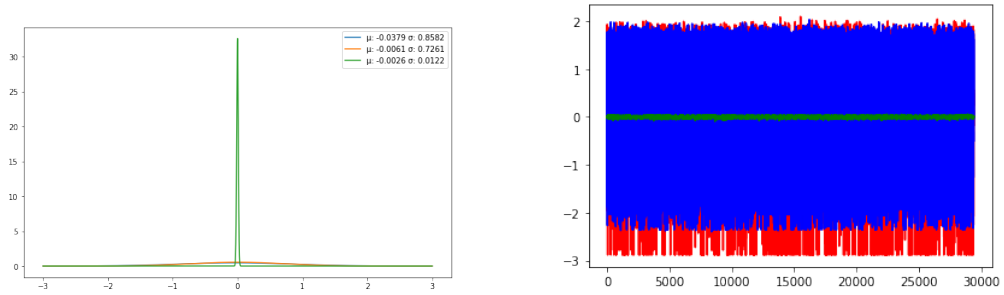


(a) Gaussians generated in case 1

(b) Plotted μ values obtained by encoder

Figure 6.3: Gaussians generated by the encoder represented as both peaks and plotted μ values

These results can be interpreted to mean that the ”wide” Gaussian is representing all 3 labels. Although the other two have converged, they seem too similar to be of any significance, it would be very difficult to distinguish between the two. Figure 6.4 shows results for a different training session with the same parameters. Results are generally, inconsistent between runs, but follow they common trend of only having **one or two** peaks which are very close to each other, and the optimizer's learning rate being **0.001** what is hereby called *Case 1*.



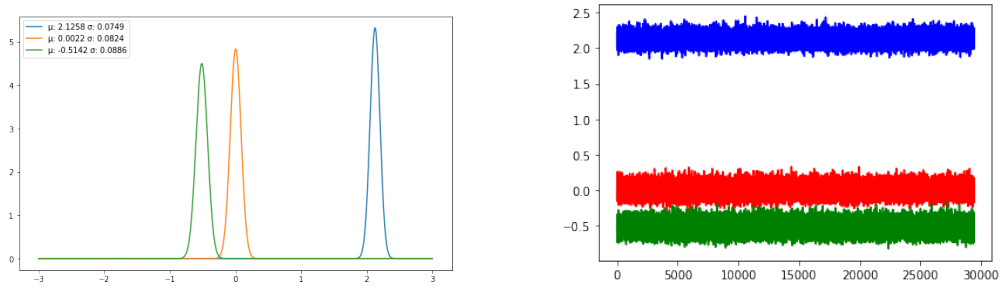
(a) Gaussians generated in Case 1

(b) Plotted μ values obtained by encoder

Figure 6.4: Gaussians generated by the encoder represented as both peaks and plotted μ values, second example

6.4.2. Case 2

By tweaking different model hyper parameters, the most interesting results came from modifying Adam optimizer's **learning rate** to **0.01**, ten times its original value, and **epsilon** to **0.001**. In this *Case 2*, results are generally more inconsistent than on *Case 1*, where the loss value occasionally tends to **infinity**. This is an indication that the learning rate is too high and is not always able to perform gradient descent. However, on some sessions the training is completed successfully with even better results than on *Case 1*. Figure 6.5 shows the encoder producing 3 distinct Gaussian peaks.

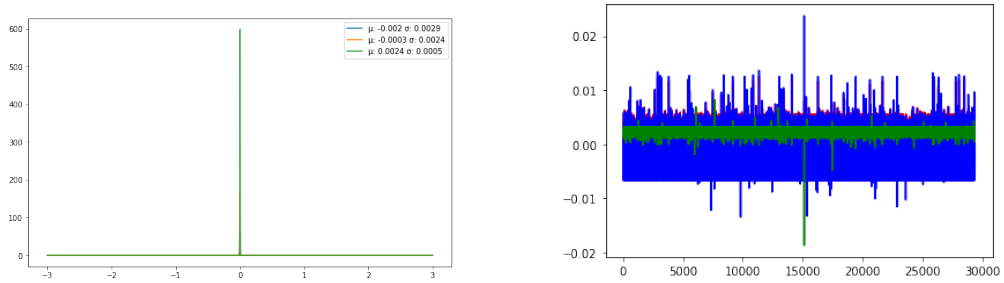


(a) Gaussians generated in Case 2

(b) Plotted μ values obtained by encoder

Figure 6.5: Gaussians generated by the encoder represented as both peaks and plotted μ values, first example

This is a strong indication that each Gaussian is representing one tag in a useful manner given that they do not overlap. These probabilistic representations are the best result to hope from these experiments. They are not particularly useful though, since it is difficult to reproduce and at the moment dependent on some luck. Figure 6.6 shows an example of a run where all 3 Gaussians have a mean of $\mu = 0$ and a standard deviation that approaches $\sigma = 0$.

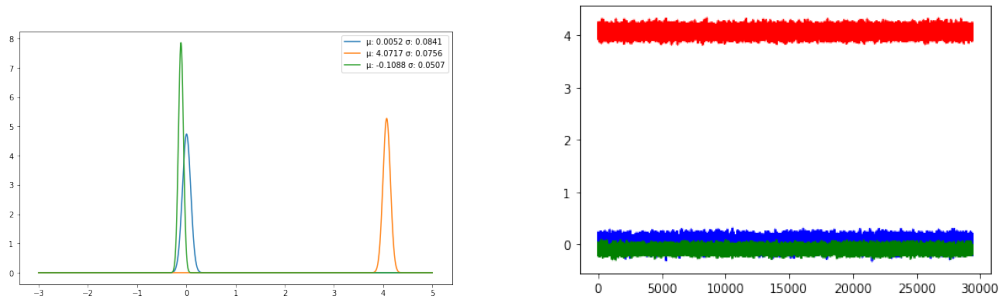


(a) Gaussians generated in Case 2

(b) Plotted μ values obtained by encoder

Figure 6.6: Gaussians generated by the encoder represented as both peaks and plotted μ values, second example

Lastly, one more interesting result can be seen in Figure 6.7, where two of the peaks overlap slightly near $\mu = 0$, but the third Gaussian has a mean value of near $\mu = 4$. Although not very reliable at this time, these results show that it is possible to train a Conditioned Variational AutoEncoder to produce Gaussian representations of a word's POS in a sentence.



(a) Gaussians generated in Case 2

(b) Plotted μ values obtained by encoder

Figure 6.7: Gaussians generated by the encoder represented as both peaks and μ values, third example

6.4.3. Case 3

In this third Case, the model is stripped from its w_k weights that are meant to select a Gaussian for the decoder of the network. This is simply to show the difference in Gaussians between the original model architecture and this modified version. The expectation is that since all 3 Gaussians would be representing the input in a combined manner, they will not peak to any value like before, and instead be quite similar. Figure 6.8 shows the new network architecture. Training the new model with the same parameters as in Case 1, yields results shown in Figure 6.9. As predicted, none of the Gaussians are peaking at any point, instead being quite similar with values close to a normal distribution $\mu = 0, \sigma = 1$. This shows that the addition of the w_k weights to the model is essential if one wants to use different Gaussians to represent the labels, as opposed to a combination of these Gaussians.

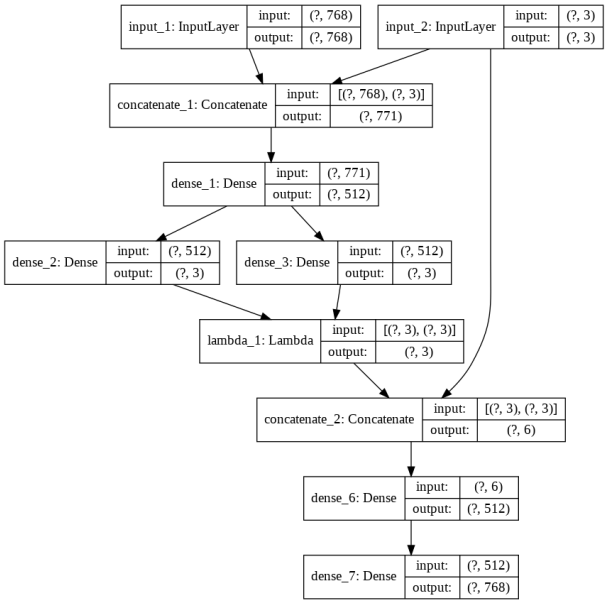
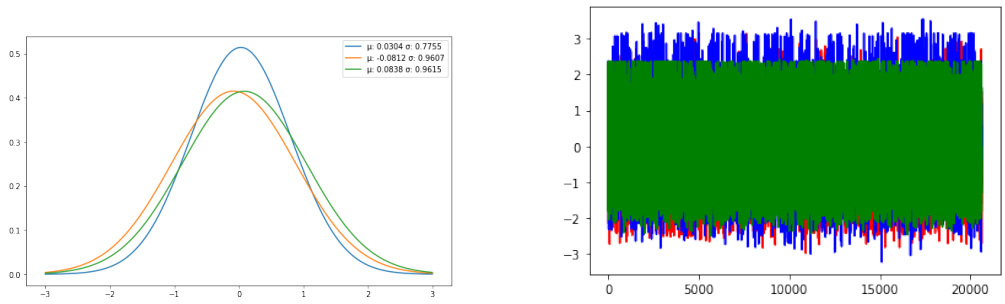


Figure 6.8: CVAE implementation architecture without w_k weights



(a) Gaussians generated in Case 3

(b) Plotted μ values obtained by encoder

Figure 6.9: Gaussians generated by the encoder represented as both peaks and μ values

7

Conclusions

BERT

To begin with, the experiments performed in Chapter 5 show that clustering methods are more effective with BERT's word contextual representations when comparing the generated clusters with the different part of speech for a particular word, as opposed to its meaning. This suggests that POS information is much easier to capture with Transformers, since it pertains to the field of **syntax**. Syntax pertains to the structure of a sentence, and as its definition suggests is usually pretty well structured, and can even be shared across multiple languages with very few differences. On the other hand, word meanings or **semantics** are much more ambiguous.

As discussed in Section 5.1.4, a single word can have many similar meanings depending wildly on the context and other nuances. There is no objective way to set a fixed amount of meanings to a word: the `time` that a person spends in prision has a specific penitentiary connotation, and could be argued to be different to the `time` defined as "the indefinite continued progress of existence and events in the past, present, and future regarded as a whole". Although in broad terms they refer to the same concept, there is an unquantifiable difference between them, even hierarchical. This does not mean BERT does not contain enough information to determine a word's meaning, only that POS information is easier to extract. This is further proven in Section 6.1 where it is shown how BERT's hidden representations can predict the POS of a word.

Gaussian representaions

Chapter 4 exhibits the theory behind a Conditioned Variational AutoEncoder model that can create probabilistic representations with different Gaussians. By conditioning BERT's hidden word representations to their corresponding POS label, Chapter 6 proves it is possible to represent each label with a different Gaussian, where Figure 6.5 is the best example. At this time however, they are not very reliable results and not trivially reproducible. Even so, the results are strong proof of concept that probabilistic representations are possible, as opposed to classical vector representations.

7.1. Future work

There are many aspects in which this work can be further improved.

Using more words

At the moment, only the word `right` has been used for the experiments for simplicity's sake. Many more words, in the order of the hundreds or thousands, could be added to the training data for the Conditioned VAE. This would dramatically increase the variety of data for the network to train on, possibly improving results.

Create probabilistic representations for word meanings

Although a much more involved task, it would be quite interesting to attempt to perform the same experiments in this document, aimed at word meanings instead of a word's POS. Word Sense Disambiguation would be the main obstacle if labels were wanted for training. It could also be interesting to let the network find the meanings in an unsupervised way, perhaps conditioning the CVAE to other BERT features, such as the sequence summary vector.

Bilingual aspect

Lastly and possibly most interestingly, this work could be expanded by not only working with one language, but two. BERT offers a multilingual model trained with hundreds of languages, which could still be used to extract word embeddings. The Gaussians for a word would come to represent different meanings, while being the same for a single meaning in two different languages. As opposed to vector embeddings, this approach could potentially better represent polysemy given its probabilistic nature.

Bibliography

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[4] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

[5] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[6] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *CoRR*, vol. abs/1802.05365, 2018.

[7] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One billion word benchmark for measuring progress in statistical language modeling," 2014.

[8] A. Radford, "Improving language understanding with unsupervised learning," Jun 2018.

[9] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[10] H. J. Levesque, E. Davis, and L. Morgenstern, "The winograd schema challenge," in *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'12, p. 552–561, AAAI Press, 2012.

[11] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.

[12] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[14] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, p. 307–392, 2019.

[15] “Stanford University School of Engineering. (2017). Lecture 13: Generative Models. [Online].” URL: https://www.youtube.com/watch?v=5WoItGTWV54. Last visited on 2021/06/16.

[16] S. Santini, “Conditioned autoencoder.” Universidad Autónoma de Madrid, 2020.

[17] I. Feinerer and K. Hornik, *wordnet: WordNet Interface*, 2020. R package version 0.1-15.

[18] M. Wallace, *Jawbone Java WordNet API*, 2007.

[19] C. Fellbaum, *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.