uc3m | Universidad Carlos III de Madrid

University Degree in Telecommunication Technologies Engineering

Academic Year 2020-2021

Bachelor Thesis

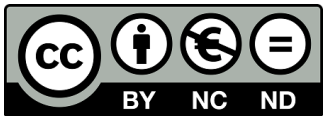
"Inside a NLP multitasking neural network. BERT, one model to rule them all"

Ion Bueno Ulacia

Supervised by:

Pablo Martinez Olmos

Leganés, June 2021



ABSTRACT

NLP is an well known topic inside AI field. A wide range of applications are collected into this area, from text classification as sentiment analysis to text generation like translation. With the potential technological growth, these tools play a critical role in daily business automating certain tasks. In spite of its benefit, NLP has been a leveraged field into AI during many years. Training times of RNNs and the difficulty to capture long-term dependencies were an extended problem. However, the emergence of the transformer architecture with its pure attention mechanisms represented a critical change in the development of models. This results in the publication of BERT, overcoming all current models and getting state-of-the-art results. This model was pre-trained and it is defined for how easy and fast can be fine-tuned for most NLP tasks. In order to show this process, two applications are developed for sentiment analysis and question answering problems. In both cases, training time was really short compared with an RNN, obtaining an outstanding performance in test set. BERT implied the origin of pre-training language models, which currently is the dominant trend in NLP area. In addition, it resulted in the solution for long training times and a huge amount of data in order to have a reliable model for a specific task.

Keywords: NLP, RNNs, self-attention, transformer, BERT, pre-training language models, fine-tuning, sentiment analysis, question answering.

ACKNOWLEDGMENTS

To my family, for her support during so many years of study. They have provided me everything in every moment to ensure I can follow the path I want.

My couple, Elena Manso. Her support and patience are limitless. She has been my pilar from the first second to the last one of the degree. Her confidence in me has been my motor during all this time.

I want to thank my friends for being the council of wise men each week and the energy refill each weekend. My classmates, partners in misfortune and the cause for many laughs during 4 years.

Finally, I am very grateful with Pablo Martinez, my thesis tutor. Thanks for his efforts and attention during the last nine months, but especially for being that professor in my first university year who I will never forget.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AI Artificial Intelligence.

BERT Bidirectional Encoder Representations from Transformers.

BPE Byte-Pair Encoding.

BPTT Back-Propagation Through Time.

BRNN Bidirectional Recurrent Neural Networks.

CMU Carnegie Mellon University.

CPU Central Processing Unit.

DRNN Deep Recurrent Neural Network.

GDD Guarantee of Digital Rights.

GDPR General Data Protection Regulation.

GLUE General Language Understanding Evaluation.

GPT Generative Pre-Trained Transformer.

GPU Graphics Processing Unit.

GRNN Gated Recurrent Neural Network.

GRU Gated Recurrent Unit.

IBM International Business Machines.

IR Information Retrieval.

LOPD Organic Law on Protection of Personal Data.

LSTM Long Short-Term Memory.

M&A Mergers and Acquisitions.

MLM Masked Language Model.

MLP Multilayer Perceptron.

NLP Natural Language Processing.

NPV Net Present Value.

NSP Next Sentence Prediction.

RACE ReAding Comprehension Dataset From Examinations.

RNN Recurrent Neural Network.

ROI Return On Investment.

RTRL Real-Time Recurrent Learning.

SQuAD Stanford Question Answering Dataset.

SST-2 Stanford Sentiment Treebank.

SWAG Situations With Adversarial Generations.

TPU Tensor Processing Unit.

UC3M University Carlos III of Madrid.

1. INTRODUCTION

1.1. Motivation of Work

As a telecommunications engineer student, the Artificial Intelligence (AI) is one branch which takes part in the contents of the degree. In spite of that, in my case there were not a significant number of courses related with the field. My learning path during the university was almost based on the fundamentals of mathematics and statistics in detection and estimation techniques. These subjects motivate me into discovering more about the field and start independent courses on my own to learn about the practical use. The contents were oriented into the bases of machine and deep learning, as well as using network services as AWS in machine learning computation.

In the last course during my stay in the Technical University of Munich, I was able to select some subjects more focused in the field. which introduce the necessary programming skills in python. I gain critical knowledge in aspects as type of models, use cases, manage of projects, programming strategies, etc. Although the most important point for me was that this experience results in the final boost to decide the topic of my bachelor thesis and the field in which I want to be specialized.

I wanted to select a relevant topic in which I could provide useful content for the current research area and whose background motivates me. Natural Language Processing (NLP) is a field which resulted me really interesting the firts time I worked on it. Looking for the last trends, I found that researches were employing NLP techniques to deal with the effects of COVID-19 pandemic. There was a wide range of use cases as covid predicion [1], [2], summarization of research articles [3], [4] or the impact in the mental health [5], [6].

Investigating in the same line, I read [7] about StructBERT [8] and how it was been employed to provide healthcare data analysis. This articles points me into the direction of understanding and working with BERT [9] due to its huge impact in the NLP community [10], [11], [12].

1.2. Objectives

The main goal consists of explaining and show how the Google's algorithm Bidirectional Encoder Representations from Transformers (BERT) [9] can be used in almost NLP task, with a great performance and a significant reduced computation time. For this purpose, making use of it we developed a couple of applications to solve two of the most common NLP tasks.

First corresponds with **sentiment analysis** whose goal is trying to extract the positive

or negative sentiment in a sentence. In spite of being a simple action, nowadays its use is really demanded and useful, for example to track the sentiment of elections [13] or Facebook's users [14].

Second one is **question answering**. As its own name indicates, the application consists of providing an answer to a question, determined by a context text which has been used as input. As before, this use and derivatives can be used in a wide range of applications as forums' automatization [15] or even in the medical context [16].

In order to provide a full and detailed explanation, it is also developed some code to provide the necessary knowledge in parts related with BERT. These ones are the **transformer** architecture and functionality, since BERT is composed by one part of it. There is also a deep explanation and demonstration of **self-attention**, the key technique used by the transformer.

1.3. Learning Outcomes

The background and contents which involve BERT imply a knowledge really distant from the base I had. Starting from a comprehensive review of the Recurrent Neural Network (RNN) and advanced derivatives. This includes their architectures and operations, as well as tehir limitations.

I was also introduced to attention mechanisms and the differences provided respect recurrence and sequential computation. Particular mention to self-attention, where I learnt its principles and how it is applied in the transformer. Consequently, one of the main learnings outcomes was how a transformer works, as well as its architecture, and the impact it involved in the NLP area.

Other important part in the learning pathway was the actual functionality of BERT, how it is the architecture and training process. This study makes me discover pre-training languages and fine-tune process, as well as its huge impact in the NLP area saving several computation time.

However, the main points were how to implement the specified tasks. It involves the management of datasets and the corresponding pre-processing per task, or even post-processing to show the results. I learnt how to load BERT using external libraries as Hugging Face [17] and fine-tune it, with an own implementation or using again Hugging Face models.

Finally mention the acquired knowledge during the use of Google Colaboratory for first time in a project. The main contribution was how to use the provided Colab's GPUs in order to save computation time.

1.4. Thesis Structure

The state-of-the-art is presented in section 2, starting with RNNs, from simplest to more advanced. It also includes attention mechanism and different architectures which were evolving until the emergence of the transformer, thanks to an optimal application of self-attention. The actual explanation and intuitive visualization of self-attention is included in section 3.

Transformer model is also presented in detail in section 4, disaggregating each of its modules. This part is critical, since it provides us the architecture employed in BERT. Training details about Google's algorithm and how to use it are explained in section 5.

Following BERT, I expose all details of the implemented applications as well as the obtained results in sentiment analysis and question answering. They correspond with sections 6 and 7 respectively.

In section 8 it is discussed some examples of subsequents models which outperform BERT. This section shows how the trend of NLP models is evolving in a short time and give us ideas on how powerful companies into AI are leading their efforts. To conclude, I commented in section 9 the obtained results and future work.

Finally, in section 10 it is analyzed the legislative and regulatory framework. The social and economic environment is presented in section 11, with a gantt chart to show the progress of the project as well as the project budget.

The code has been developed in Python 3.8 with Colab to develop the models and PyCharm to test external code. I have also used Hugging Face models [17] and datasets [18] in the tasks' implementation.

Mention that the memory has been written in parallel with the implementation of the Colab notebook, so some parts are identical. I published in my github (github.com/ion-bueno/bert-from-inside) the notebook (BERT, one model to rule them all) with all the explanations and the option to run it in Colab. It is also uploaded my implemented models and results.

2. STATE OF THE ART

As it is defined in [19], Natural Language Processing (NLP) studies how computers can understand and manipulate natural languages to perform certain tasks. Multiple applications can be derived from this area, such as text classification, text modelling, translation, questions and asnwering, summarization, between others.

The origin dates from 1950s, where NLP and Information Retrieval (IR) [20] were working areas quite different. Currently, different experts in fields work together to broaden the general knowledge in the area [21].

Natural Language Processing (NLP) encloses any processing or generation of text. It is an example of supervised learning, where in a simple view, there is a text which is passed to a model which generates a certain outcome or prediction [22].

$$\underbrace{f(\overbrace{x}^{\text{input}})}_{\text{model}} \approx \underbrace{y}_{\text{outcome/prediction}} \tag{2.1}$$

One problem with the text is that documents are of variable length. This requires a transformation into a fixed size vector [22].

$$f : \underbrace{\mathbb{R}^d}_{\text{fixed size vector}} \rightarrow \mathbb{R} \tag{2.2}$$

One classic way is **bag of words** [23], where there is one dimension per word in vocabulary. This approach leads to having almost all values zeros and consequently an innefficient memory use.

In addition to that, another important aspect is the order of words. Different positions of the words in same sentence can lead to multiple meaning combinations. One proposal to deal with it is **N-grams**. The goal is to capture the meaning of text and do not be affected by the positions [24]. In contrast, this algorithm involves an increase in the dimensionality of vectors due to the multiple possibilities [22]. There are already implementations which are improving the mentioned algorithm [25].

Text is an example of sequence data. This means samples are not independent between each other and they could not be identically distributed. This completely differs from images, where models are training to look for regularity and spatial data. **Recurrent Neural Networks (RNNs)** are designed to handle sequential data [26], becoming the classical approach until 4 years ago.

2.1. Recurrent Neural Networks (RNNS)

Recurrent neural networks (RNNs) were declared as state of the art in sequence modelling and transduction problems [27]. There are several examples as language modelling [28] [29], paraphrising [30], entity recognition [31], translation [32], [33], between others. In addition to processing or generating text, they have been used in other applications as computer vision [34], [35].

A RNN can be defined as an extension of a conventional feedforward neural network [36] plus a recurrent hidden state. These networks uses recurrent computation for hidden states, that is the reason why they are called recurrent neural network [26].

The activation of this special layer is dependent of previous time. They generate a sequence of hidden states h_t, function of previous hidden states h_{t-1} and current input x_t [26], [37].

$$h_t = f(x_t, h_{t-1}) \tag{2.3}$$

If we consider the samples are not independent between them, the sequence probability $x = (x_1, x_2, \ldots, x_N)$ can be decomposed into

$$p(x_1, \ldots, x_N) = p(x_1)p(x_2|x_1)\ldots(x_N|x_1, \ldots, x_{N-1}) \tag{2.4}$$

where each conditional probability is equal to

$$p(x_t|x_1, \ldots, x_{t-1}) = \phi(h_t) \tag{2.5}$$

ϕ is a a smooth bounded function, logistic sigmoid or hyperbolic tangent function.

Following the explanation given in [26] it is introduced the hidden state in order to understand how a RNN works. Given a minibatch $X_t \in \mathbb{R}^{nxd}$ with batch size n and d inputs at time step t, the hidden state $H_t \in \mathbb{R}^{nxh}$ with h hidden units is calculated as

$$H_t = \phi(X_t W_{xh} + H_{t-1} W_{hh} + b_h) \tag{2.6}$$

where $W_{xh} \in \mathbb{R}^{dxh}$, $W_{hh} \in \mathbb{R}^{hxh}$ are weight matrices and $b_h \in \mathbb{R}^{1xh}$ the bias.

Finally, the output in the output layer corresponds with:

$$O_t = H_t W_{hq} + b_q \tag{2.7}$$

being again $W_{hq} \in \mathbb{R}^{hxq}$ a weight matrix and $b_q \in \mathbb{R}^{1xq}$ the bias term.

Hidden state allows capturing historical information of the sequence until current time step. Besides that, the number of parameters does not grow as the number of time steps increases [26]. Due to these features, RNN have been the predefined architecture to use in NLP tasks.

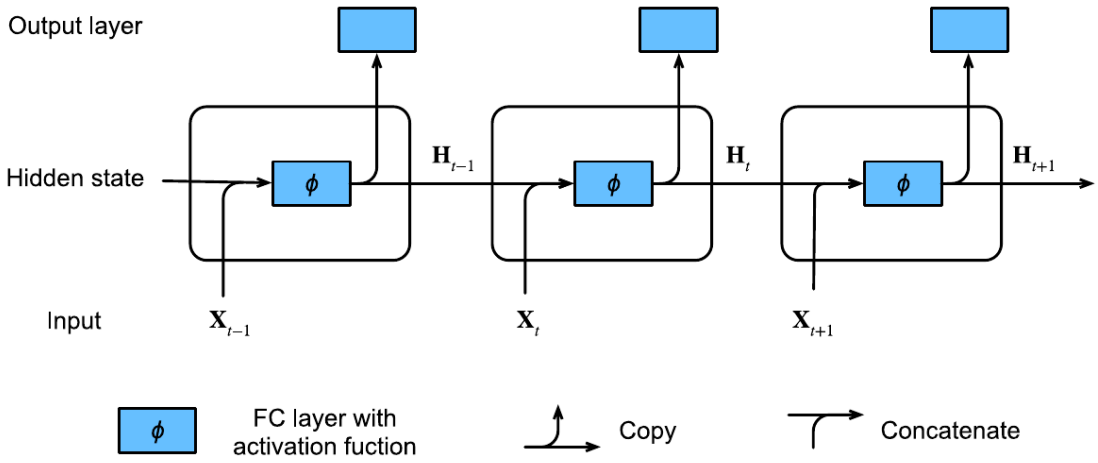


Fig. 2.1. RNN architecture [26].

2.1.1. Problems with RNNs

In contast, RNNs present some significant drawbacks. First, training time takes long time [38]. Optimized algorithms as Back-Propagation Through Time (BPTT) or Real-Time Recurrent Learning (RTRL) [39] are very slow with long-term sequences. Truncated BPPT [40] was designed as a solution, under the idea of using only short-term dependencies. In spite of that, this assumption is declined in advanced RNN.

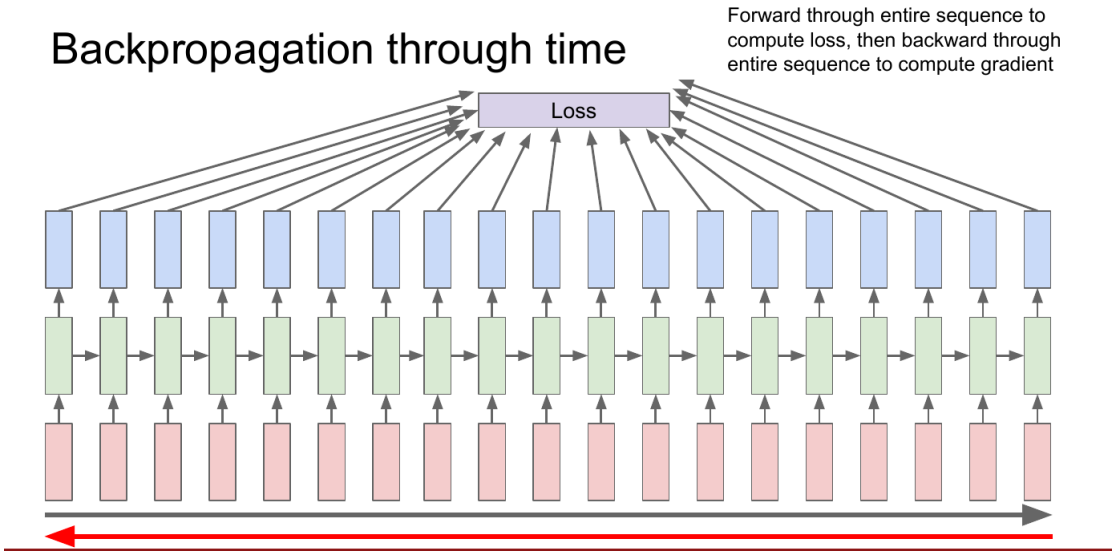


Fig. 2.2. Intuition about BPTT [41].

Another important issue is the difficulty to capture long-term dependencies by RNNs. Gradient based learning algorithms face many problems as the sequence or spans increases [42]. Then error signals used during backpropagation could [38]:

1. Tend to blow up, leading to oscillating weights.

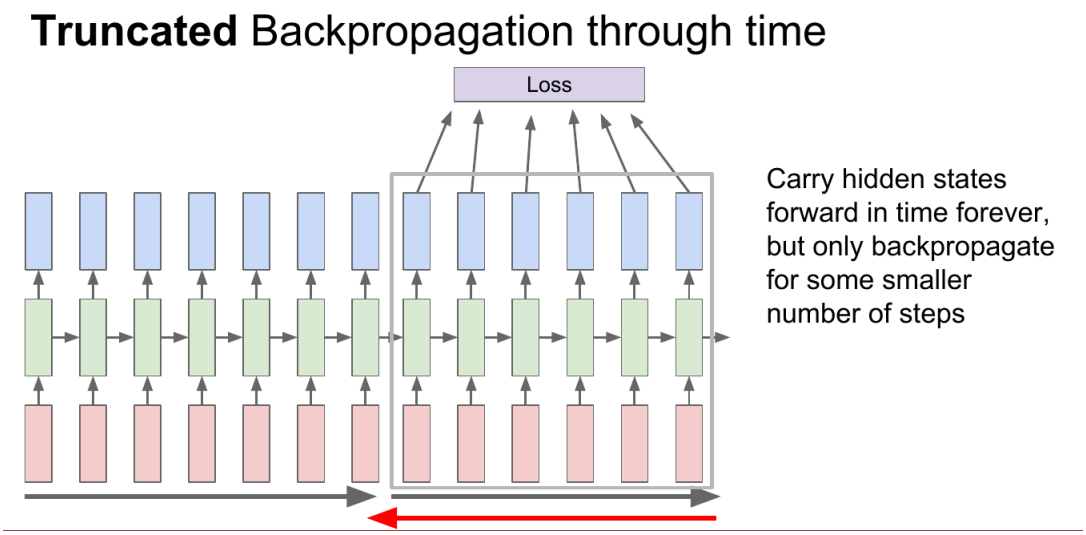


Fig. 2.3. Intuition about Truncated BPTT [41].

2. Vanish, causing extreme training times.

These effects do not ease the gradient optimization, due to the highly variations. Besides that, the effect of long-term dependices is completely absorbed by short-term ones [37].

To deal with it different approaches were cosidered. First one encloses advances in optimizing RNNs [43] or utilizing Hessian-free optimization approach [44] [45].

Second one resulted as the next solution for NLP tasks. It is based in the design of a more sophisticated activation function. The result were the introduction of the **Long Short-Term Memory (LSTM)** [46] and **Gated Recurrent Unit (GRU)** [47], which perform well capturing long-term dependencies in tasks as speech recognition [48] or translation [32].

2.2. Gated Recurrent Neural Networks (GRNNs)

As it has been discussed, long products of matrices in RNNs can lead to vanishing or exploding gradients. There are many cases where this results critical. For example there might be a situation where some token is vital for the sequence, requiring a mechanism to store the information. Otherwise, some info could not have any value and skipping it is the best way. Also, a strategy to differentiate between parts of a sequence is needed in some situations [26].

Between the proposed methods, LSTM and GRU stand out. LSTM performs better in general terms, altough GRU, with a simpler architecture, offers comparable performance and it is faster [37].

2.2.1. Long Short-Term Memory (LSTM)

Presented first time in [46] and almost two decades before GRU, altough with a more complex design. LSTM was designed to overcome the problems given in the backpropagation process. This is achieved by a gradient-based algorithm for an architecture enforcing constant error flow through internal states of special units [46].

LSTM is inspired by logic gates of a computer [26]. To get a fast intuition, it can be seen a network with different switches which decide if the information pass or not [37].

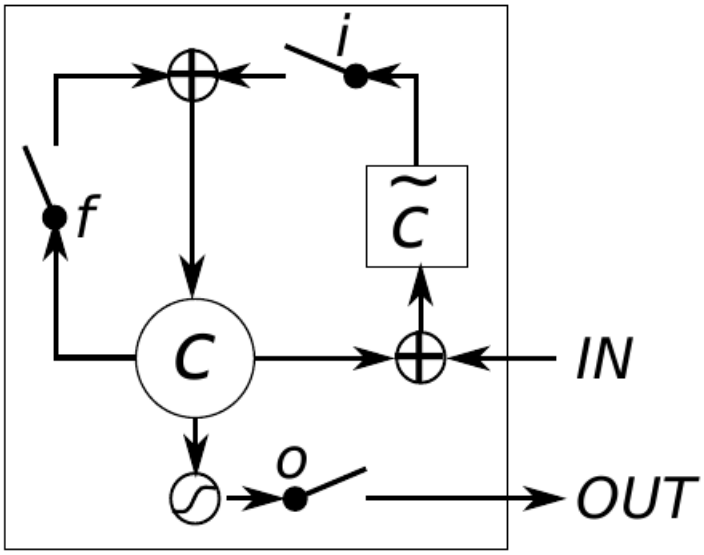


Fig. 2.4. LSTM architecture [37].

To give an introduction about its functionality, it is going to be followed the explanation in [26].

Memory Cell

It is introduced a **memory cell** with same size than hidden state to record additional information. This one is compounded by an **output gate** to read out the entries from the cell. Additionally, it is needed the **input gate** to decide when to read data into the cell. Finally, the **forget gate** is in charge of reset the content of the cell. The values of the three cells are in the range of (0,1).

If there are h hidden units, n is the batch size and d the number of inputs. It is defined the input $X_t \in \mathbb{R}^{nxd}$ and the hidden state of the previous time step $H_{t-1} \in \mathbb{R}^{nxh}$. Then it is presented next mathematical equations for the **input gate** $I_t \in \mathbb{R}^{nxh}$, **forget gate** $F_t \in \mathbb{R}^{nxh}$

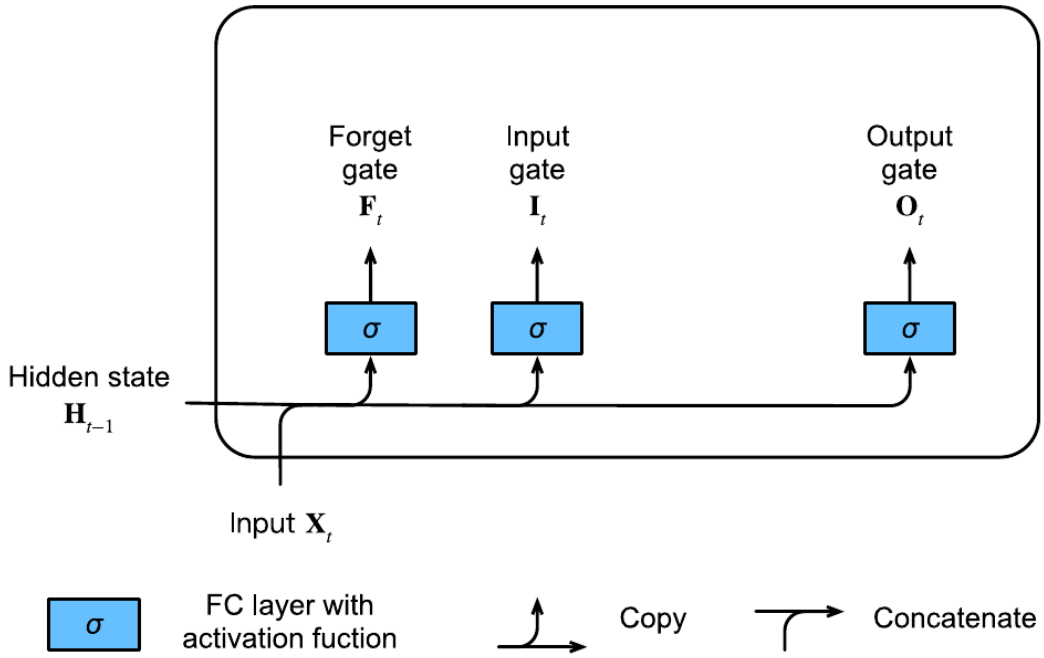


Fig. 2.5. Gates in LSTM memory cell [26].

and the **output gate** $O_t \in \mathbb{R}^{nxh}$:

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i) \tag{2.8}$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f) \tag{2.9}$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o) \tag{2.10}$$

being W_{xi}, W_{xf}, $W_{xo} \in \mathbb{R}^{dxh}$ and W_{hi}, W_{hf}, $W_{ho} \in \mathbb{R}^{hxh}$ weight parameters and b_i, b_f, b_o $\in \mathbb{R}^{1xh}$ bias parameters.

In order to design the memory cell, the **candidate memory** is introduced. $\tilde{C}_t \in \mathbb{R}^{nxh}$ follows a similar equation than other gates, with the detail of using tanh as activation function:

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c) \tag{2.11}$$

where again $W_{xc} \in \mathbb{R}^{dxh}$ and $W_{hc} \in \mathbb{R}^{hxh}$ are weight parameters and $b_c \in \mathbb{R}^{1xh}$ bias parameter.

The input gate I_t manages how much significant is the new data through $\tilde{C}_t$, while the forget gate F_t is in charge of determine how much of the old memory cell content $C_{t-1} \in \mathbb{R}^{nxh}$ should be retained. Using pointwise multiplication ($\odot$), cell moment content is defined as:

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t \tag{2.12}$$

The forget gate determines if the past memory cells C_{t-1} is saved and passed to the current time step, as well as the input gate with the previous hidden state H_{t-1} presented in the

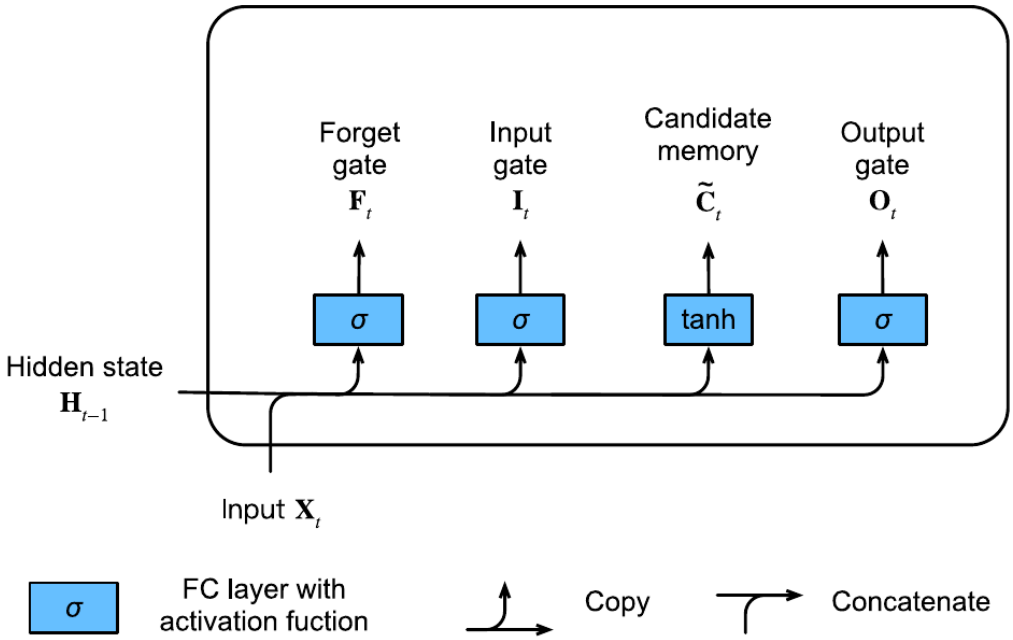


Fig. 2.6. Candidate memory in LSTM [26].

caditate memory $\tilde{C}_t$. This implementation alleviates the vanishing gradient problem and helps to capture better long range dependencies.

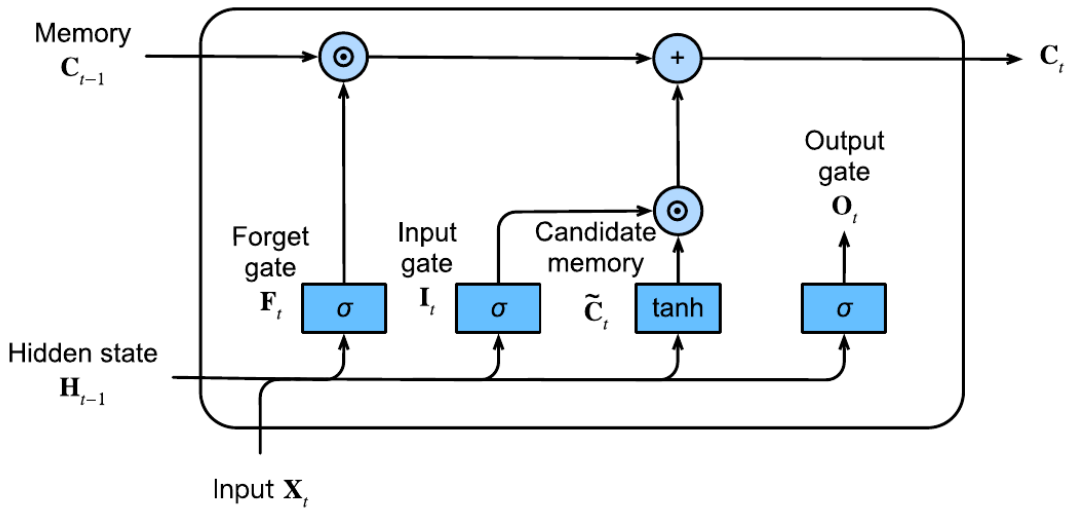


Fig. 2.7. Memory cell computation in LSTM [26].

Hidden State

To compute the hidden state $H_t \in \mathbb{R}^{n \times h}$ it is employed the output gate O_t. Using tanh as activation function, it is declared:

$$H_t = O_t \odot \tanh(C_t) \tag{2.13}$$

As it can be seen, here the output gate O_t is the one which decides to pass the memory information through to the predictor or retain it within the memory cell.

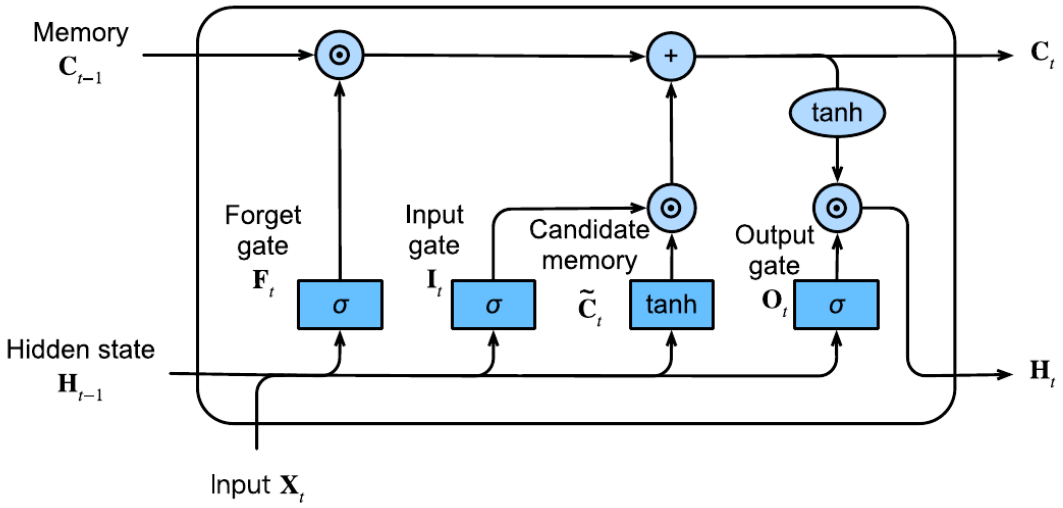


Fig. 2.8. Hidden state computation in LSTM [26].

Thanks to the introduced gates, LSTM is able to decide whether to keep the exisiting memory. If it is detected an important feature very early, this information is easily carried over a long distance capturing long-term dependencies [37].

2.2.2. Gated Recurrent Unit (GRU)

Developed in [47], GRU often offers a comparable performance respect LSTM. Its design is simpler, dedicating mechanisms for when hidden state should be updated or reset. Each recurrent unit is in charge of capturing dependencies of different time scales. There are gating units which modulate the flow or information. The difference is there are not separate memory cells [37]. The values of the cells are also in range (0,1).

The model works similar than LSTM, therefore the functionality can be shown as a circuit with switches.

As before, it is followed the explanation given in [26].

Reset Gate and Update Gate

Having an input $X_t \in \mathbb{R}^{nxd}$ and the hidden state of the previous time step $H_{t-1} \in \mathbb{R}^{nxh}$, **reset gate** $R_t \in \mathbb{R}^{nxh}$ and **update gate** $Z_t \in \mathbb{R}^{nxh}$ are defined as:

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r) \tag{2.14}$$

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z) \tag{2.15}$$

being W_{xr}, $W_{xz} \in \mathbb{R}^{dxh}$ and W_{hr}, $W_{hz} \in \mathbb{R}^{hxh}$ weigth parameters and b_r, $b_z \in \mathbb{R}^{1xh}$ biases.

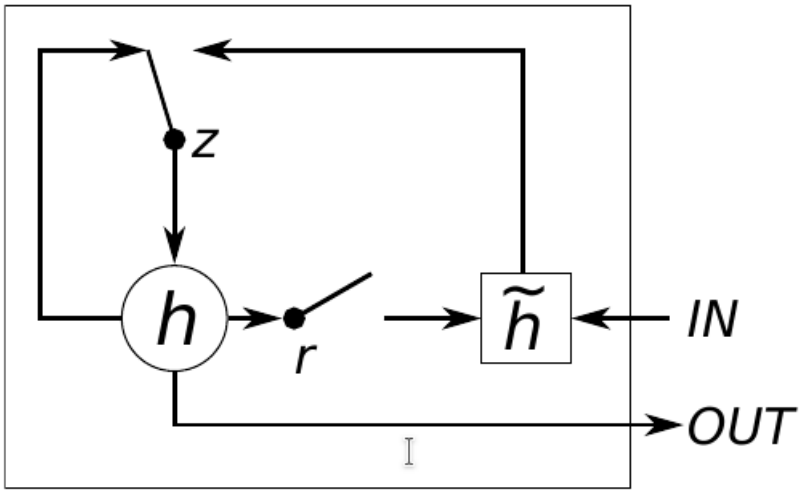


Fig. 2.9. GRU architecture [37].

Reset gates help capture short-term dependencies while update gate is in charge of long-term dependencies.

Hidden State

In order to get the hidden state, it is used the reset gate. The idea is generate a **candidate hidden state** $\tilde{H}_t \in \mathbb{R}^{nxh}$.

$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h) \tag{2.16}$$

where $W_{xh} \in \mathbb{R}^{dxh}$ and $W_{hh} \in \mathbb{R}^{hxh}$ weight parameters and $b_h \in \mathbb{R}^{1xh}$ the bias.

If entries in reset gate R_t are close to 1, the model is a standard or vanilla RNN [49], while if they are close to 0 the result is a Multilayer Perceptron (MLP) [36] with X_t as input.

Once the candidate hidden state is generated, it is used the forget gate to finally update the hidden state:

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t \tag{2.17}$$

In this case if the update gate Z_t is close to 1, the old state is retained and X_t ignored. In contrast, if Z_t is close to 0, H_t approaches to the candidate latent state $\tilde{H}_t$

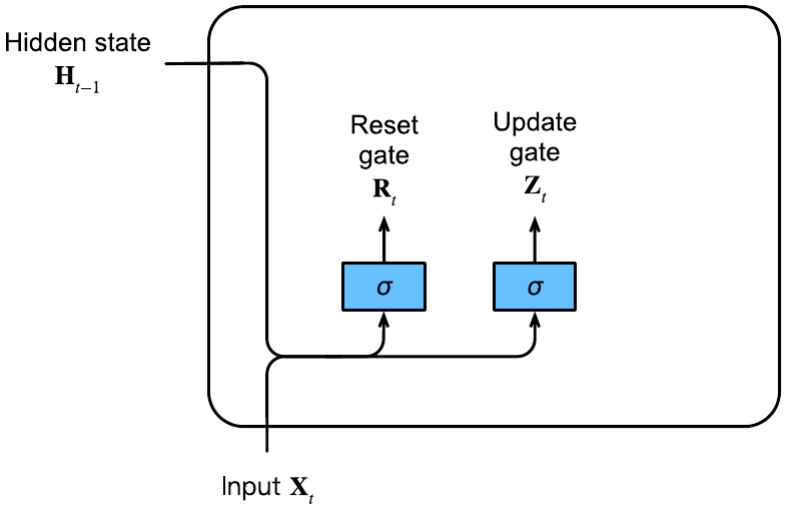


Fig. 2.10. Gates in GRU [26].

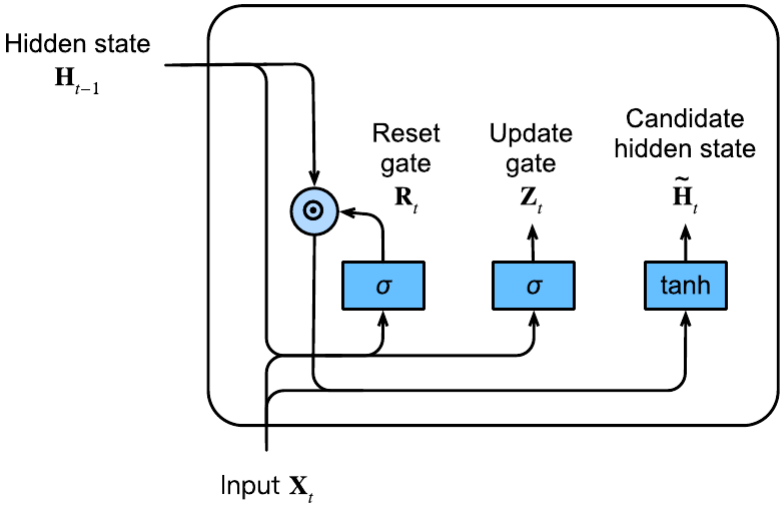


Fig. 2.11. Candidate hidden state in GRU [26].

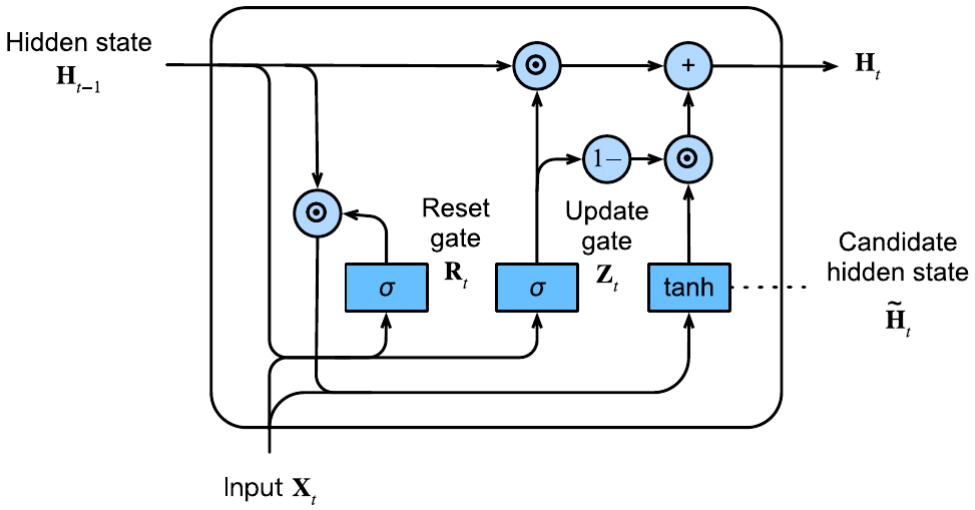


Fig. 2.12. Hidden state in GRU [26].

One feature shared between both activation functions is the additive component of their update. It differs from traditional RNN, which replaces the content of a unit with a new value computed from the current input and the previous hidden state. In contrast, LSTM and GRU keep the existing content and add the new information. This addition involves clear advantages. First, each unit easily remembers a feature in the input stream during a long sequence. Second, the error is easily back-propagated without vanishing due to shortcut paths created by this addition [37].

Both activations have been evaluated and compared with the performance of a traditional tanh unit. Gated units clearly shown a superiority, altough without a concrete conclusion which of both is better [37].

2.2.3. Problems with GRNNs

In spite of presenting clear advantages, GRNNs also present critical drawbacks, which have been a research motivation.

Mainly, these gates requires so much time to train. Their designs do not fit with the parallelization strategies and technologies developed to reduce computation time [50]. That is also due to the gradient paths that are extended as the sequence increases. For example, a net based on LSTM which is going to process sequences of 100 words or tokens is similar to a MLP with 100 layers [22].

In addition to the long times, transfer learning (learning a new task through the transfer of knowledge from a related task than has been learned [51]) really does not work in these implementations. Also mention it is needed a specific labeled dataset for every task [22].

Many efforts have been carried out to push the boundaries of RNN and to find different strategies which can deal with the mentioned problems [27].

2.3. Deep Recurrent Neural Networks (DRNNs)

How latent variables and observations interact is not very intuitive. Into practical effects this does not result into a problem if there is enough flexibility to model the different types of interactions. Nevertheless, with one single layer it could be difficult [26].

One solution is given by using a Deep Recurrent Neural Network (DRNN). One point to remark is the concept of depth in an RNN is not as clear as in a MLP. Different strategies are evaluated to find in which points a RNN may be made deeper [52].

For this explanation, it is going to be supossed a RNN with multiple hidden layers as in [26].

There is not a big difference respect the equation 2.6, it is defined the hidden state respect layer

$$H_t^{(l)} = \phi_l(H_t^{(l-1)})W_{xh}^{(l)} + H_{t-1}^{(l)} + b_h^{(l)} \tag{2.18}$$

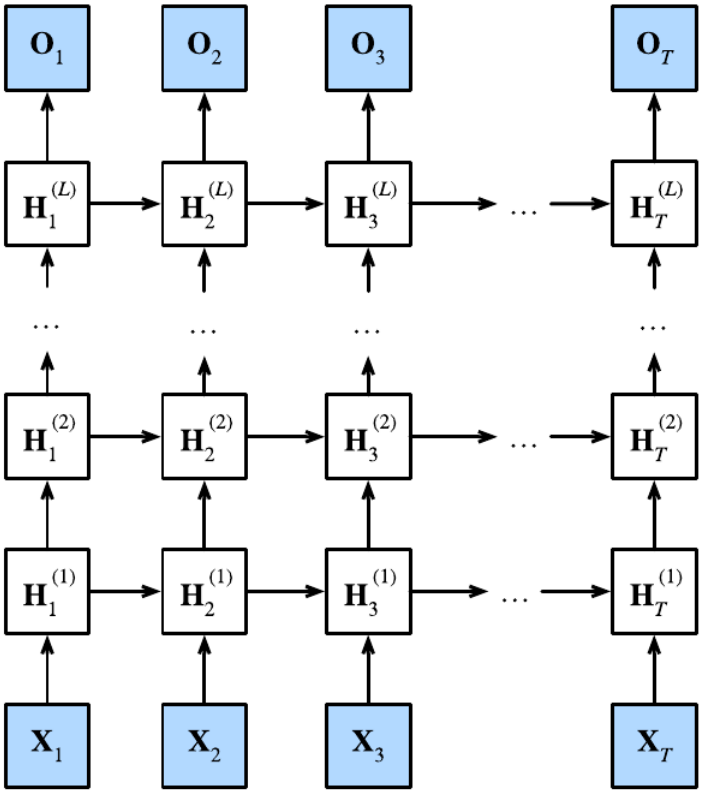


Fig. 2.13. Deep RNN [26].

The output is only based on the final L^{th} hidden layer, similar than equation 2.7

$$O_t = H_t^{(L)} W_{hq} + b_q \tag{2.19}$$

As with an standard MLP network [36], the number of hidden layers L and the number of hidden units h are hyperparameters. Mention the calculation of the hidden state in equation 2.6 can be replaced by a LSTM [46] or GRU [47] cell.

Deep RNNs have been widely used in many tasks. Some of them are speech recognition [53], image classification [54], video translation [55], between others.

2.4. Bidirectional Recurrent Neural Networks (BRNNs)

Proposed in [56] and in [57] with the goal of predicting the secondary structure of a protein. This type of RNN is trained simultaneously in positive and negative time direction. These networks outperform unidirectional ones, being more faster and accurate [58].

Intuitively, the equations do not differ from 2.6. Now, there is the forward hidden state $H_t^{(f)}$ and the backward hidden state $H_t^{(b)}$. Following [26] explanation, the hidden states

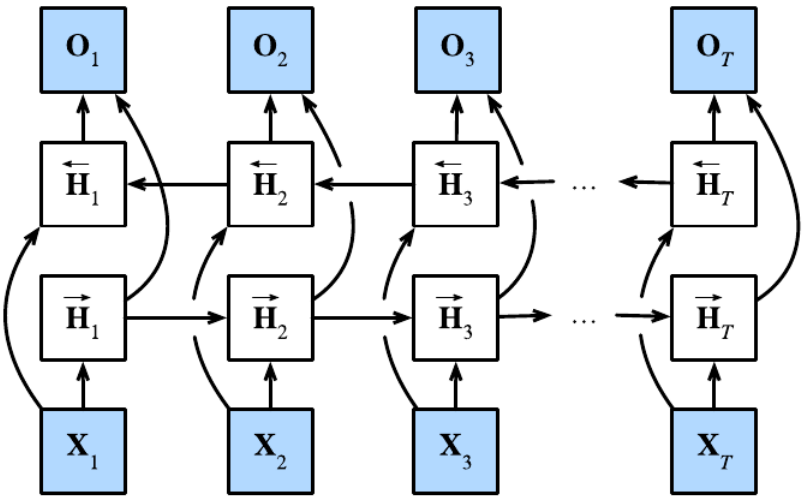


Fig. 2.14. Bidirectional RNN [26].

are defined as

$$H_t^{(f)} = \phi(X_t W_{xh}^{(f)} + H_{t-1}^{(f)} W_{hh}^{(f)} + b_h^{(f)}) \tag{2.20}$$

$$H_t^{(b)} = \phi(X_t W_{xh}^{(b)} + H_{t+1}^{(b)} W_{hh}^{(b)} + b_h^{(b)}) \tag{2.21}$$

The forward and backward hidden states are concatenated into $H_t \in \mathbb{R}^{nx2h}$ and passed into the output layer. The output is computed in the same way as in equation 2.7

$$O_t = H_t W_{hq} + b_q \tag{2.22}$$

with the weight matrix $W_{hq} \in \mathbb{R}^{2hxq}$ and the bias $b_q \in \mathbb{R}^{1xq}$, being q the number of outputs. Mention the two directions can have different number of hidden units.

BRNNs have given improved results in sequence learning tasks [58]. Some to highlight are protein structure prediction [59], [60] and speech processing [61], [62].

2.5. Encoder-Decoder Architecture

Transduction problems such as language modelling and machine translation are very common [29], [32] in NLP area. In this type of problems the input and output are both variable-length sequences. The encoder-decoder architecture was designed for this purpose [26].

The **encoder** takes a variable-length sequence as the input and transforms it into a state with a fixed shape. Then, the **decoder** maps the encoded state to a variable-length sequence [26].

Fig. 2.15. Encoder-Decoder architecture [26].

2.5.1. Sequence to Sequence Learning

In the case of sequence learning, the input sequence is encoded and then next token is predicted by the decoder. The model stops making predictions once the special token "<eos>" is generated [26]. The beginning of sequence is usually indicated by "<bos>".

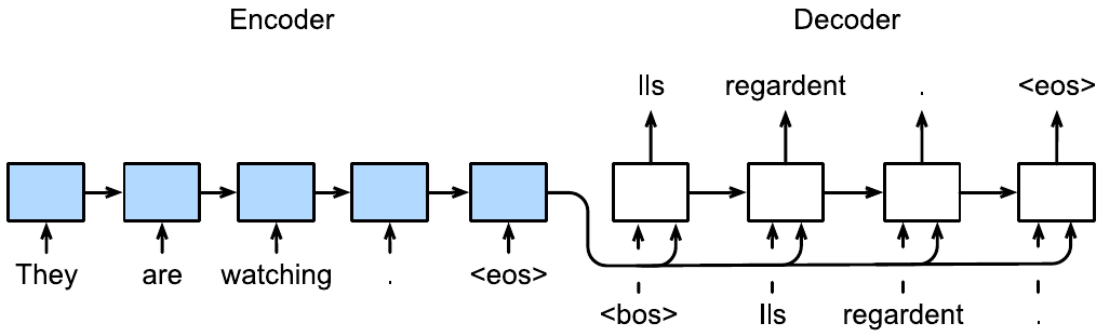


Fig. 2.16. Sequence to sequence learning with a encoder-decoder architecture [26].

There are different strategies respect the initialization of the decoder. In [32], the final hidden state of the encoder is also fed into the decoder as part of the inputs. The encoder and the decoder are jointly trained to maximize the conditional probability of a target seuqence given a source sequence. It is shown the performance improves by using the conditional probabilities of phrase pairs as an additional feautre in the model.

In contrast, in [29] the final hidden state of the encoder is used to initiate the hidden state of the decoder. This architecture is composed by a LSTM as encoder and a deep LSTM as decoder.

There are several examples [63], [64] in machine translation which use this type of architecture and obtained state of the art results.

2.6. Optimized Recurrent Neural Networks

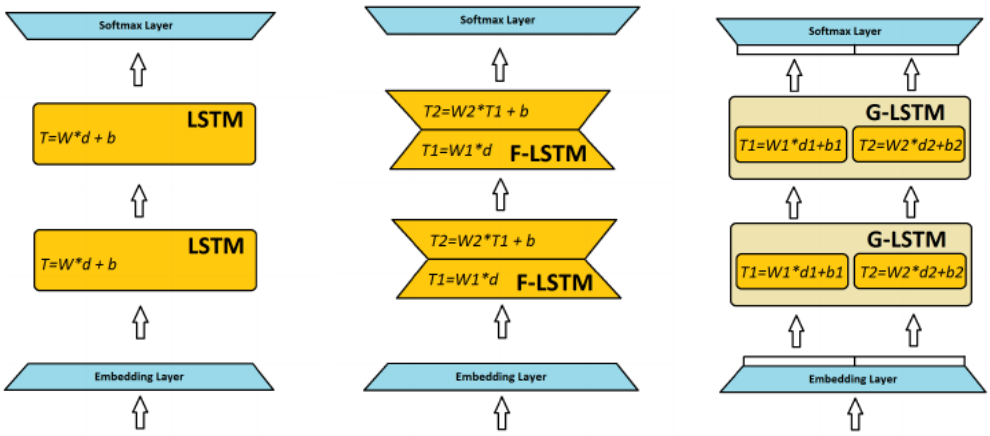
Sequence data is clearly a problem respect computation times. The topic encloses a large list of models and strategies to deal wih it. There have been recent improvemenents in computational efficiency.

2.6.1. Factorization Tricks

Two models to highlight are factorized LSTM and group LSTM presented in [65].

The **factorized LSTM (F-LSTM)** replaces a big weight matrix W by a product of two smaller matrices $W_1 \cdot W_2$. This approximation contains less parameters and the computation is faster in case os distributed training. The idea of factorization was already used in VGG networks [66] and ResNet [67].

Group LSTM (G-LSTM) consists in splitting LSTM cell into independent groups. The idea is based in that some parts of the input x_t and hidden state h_t can be presented as independent feature groups. The approach is inspired by [68], where some convolutional layers are divided into two groups in order to split the model between two GPUs. This strategy is followed in other models as convolutional networks [69] or Xception [70].



(a) Regular LSTM (b) Factorized LSTM (c) Group LSTM

Fig. 2.17. Comparation of the three different LSTM cells [65].

2.6.2. Conditional Computation

Another advance to remark is conditional computation. This strategy, where parts of the network are active or inactive on a per-example basis, has been proposed in several models [71], [72] [73] without a proportional increase in computational costs.

In [74] is introduced a new general purpose neural network component, **Moe**. This type consists in multiples feed-forward neural networks, called experts, and a gating network which selects a combination of intra nets to process each input. The differents experts tend to become specialized based on syntax and semantics.

Experts were introduced in [75], [76], so different architectures has been proposed as SVMs nets [77], hierarchical experts structure [78] or sequence of experts [79].

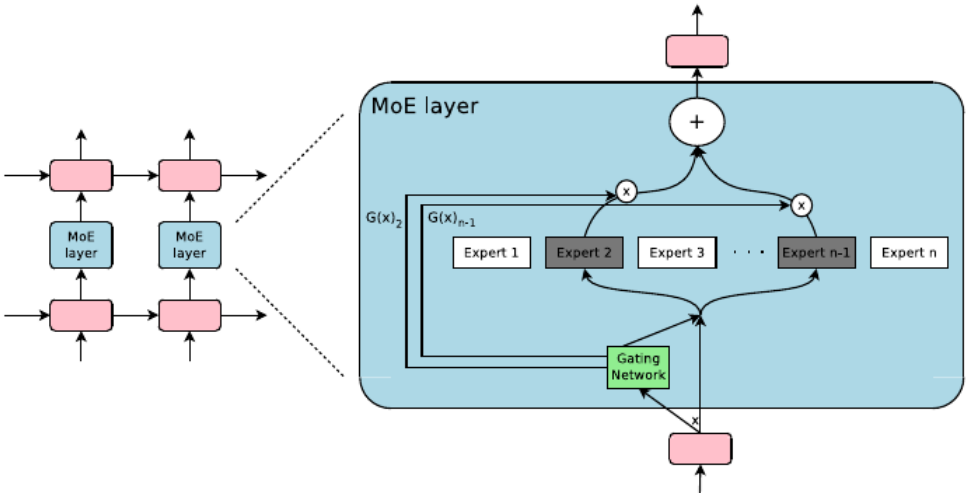


Fig. 2.18. Cobination of experts and a gating network (MoE) [74].

It has been mentioned sections 2.1.1 and 2.2.3, the fundamental constraints of these architectures. How RNNs' designs imply long training times and exploit or vanishing gradients. In the case of sequence-to-sequence learning, the encoder-decoder architecture is not able to accurately process long input sequences, since the decoder is only fed with the last hidden state of the encoder [80].

In spite of some improvements (section 2.6) in sequential computation, it still results a problem in efficiency. The solution designed to definitely deal with this issue is **attention**.

2.7. Attention Mechanisms

Attention was originally introduced to solve the issues about sequence-to-sequence models. It achieved considered improvements in machine translation [63], [33], [64].

The idea comes from the theory of the *feature integration theory* of the selective attention in [81]. It declares than when perceiving a stimulus, features are registered early and in parallel, while objects are identified separately in a later stage. For neural networks, attention can be seen as a generalized pooling method, to retain the most significant information [82], with bias alignment over inputs [26].

Following the explanation given in [26], the core component is the **attention layer**. The input is called **query**, for which the layer returns a set of **key-value** pairs encoded in the attention layer based on the memory.

Let's assume there are n key-value pairs $(k_1, v_1), \ldots, (k_n, v_n)$ with $k_i \in \mathbb{R}^{d_k}$, $v_i \in \mathbb{R}^{d_v}$. Given a query $q \in \mathbb{R}$, it is computed a score function α which measures the similarity between the query and the key. For each key $k_1, \ldots, k_n$ the scores are $a_1, \ldots, a_n$

$$a_i = \alpha(q, k_i) \tag{2.23}$$

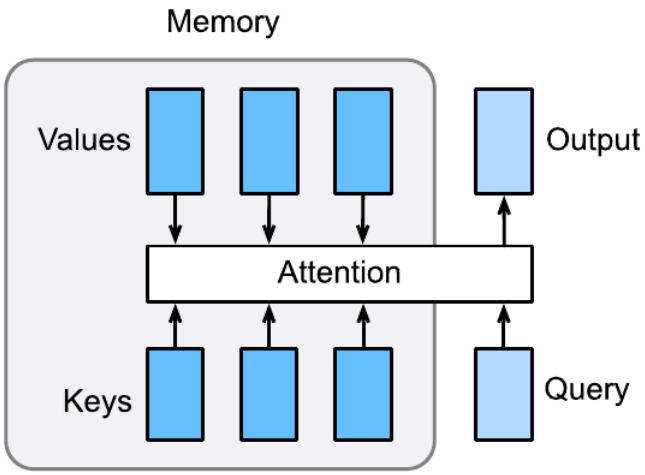


Fig. 2.19. Attention layer diagram [26].

To get the attention weigts, it is applied a softmax.

$$b_i = \frac{\exp^{a_i}}{\sum_j \exp^{a_j}} \tag{2.24}$$

The attention layers return an output $o \in \mathbb{R}^{d_v}$ with the same shape as the value. This corresponds with a weighted sum of the values:

$$o = \sum_{i=1}^{n} b_i v_i \tag{2.25}$$

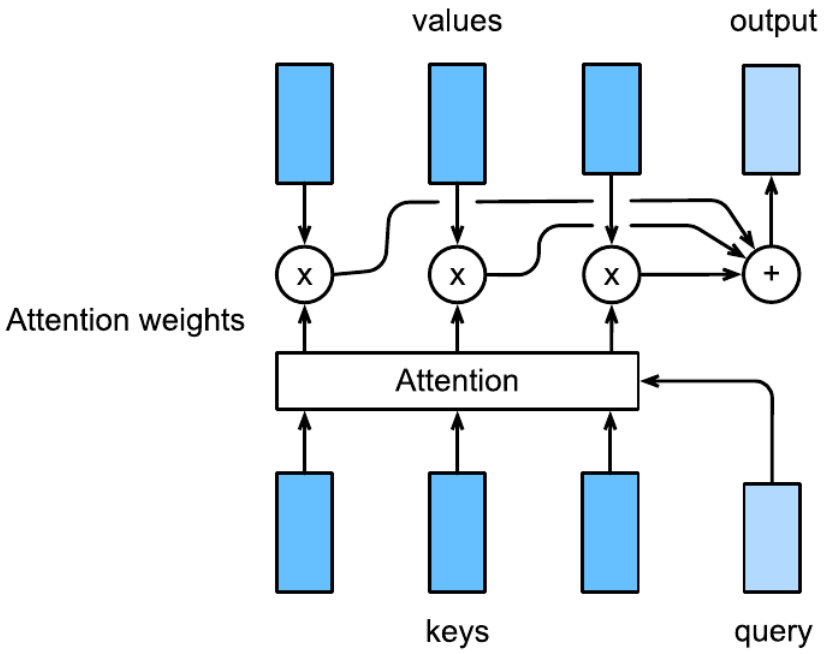


Fig. 2.20. Attention output [26].

2.7.1. Types of Attention

Mainly, there are two types of attention [80], **Bahdanau** and **Luong**. The principles are the same in both types, while they differ in the architecture and computations.

Bahdanau Attention

Introduced in [33] and referred as **additive attention**. The paper describes a model which automatically search parts of a source sentence that are relevant to predicting a target word, avoiding the use of a hard segment. The idea is based in the conjecture that the use of a fixed-length vector get a worse performance of the encoder-decoder architecture.

The model uses a combination of RNN plus attention mechanism. In the original paper [33] they use a GRU (section 2.2.2) as activation function for the RNN.

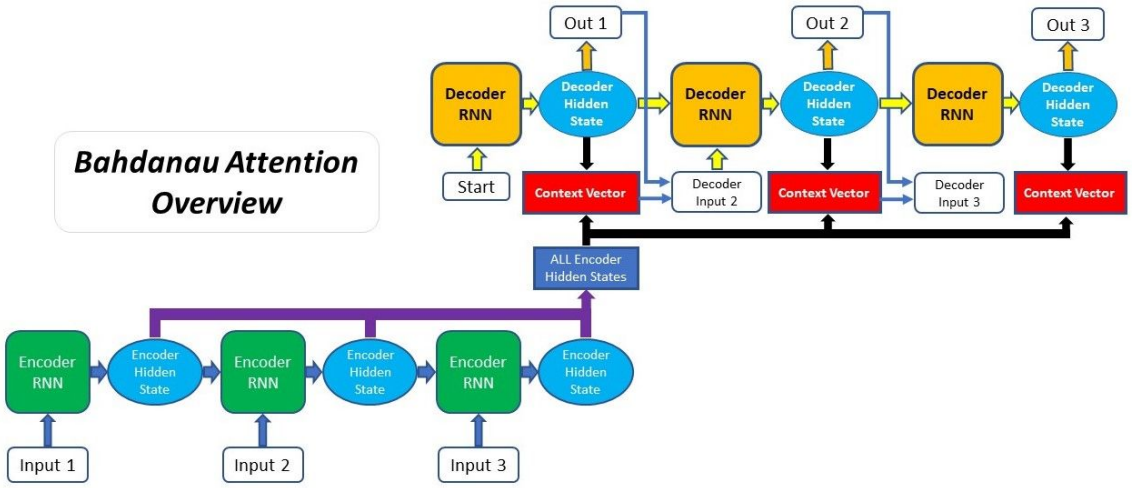


Fig. 2.21. Bahdanau attention in sequence to sequence model [33].

To calculate the scores, they use a **MLP attention** [26] to project both query and keys into $\mathbb{R}^h$ by learnable weights parameters.

$$\alpha(k, q) = v^T \tanh(W_k h + W_q q) \tag{2.26}$$

As it can be seen, the activation function is tanh and there is not bias.

Luong Attention

Presented in [64], it is called **multiplicative attention**. The main difference with Bahdanau attention (2.7.1) is the position at which the attention mechanism is being introduced in the decoder. While in the previous type the context vector is only utilised after RNN and produces the output for that time step, now it is combined with the decoder hidden state to generate the final output for that word.

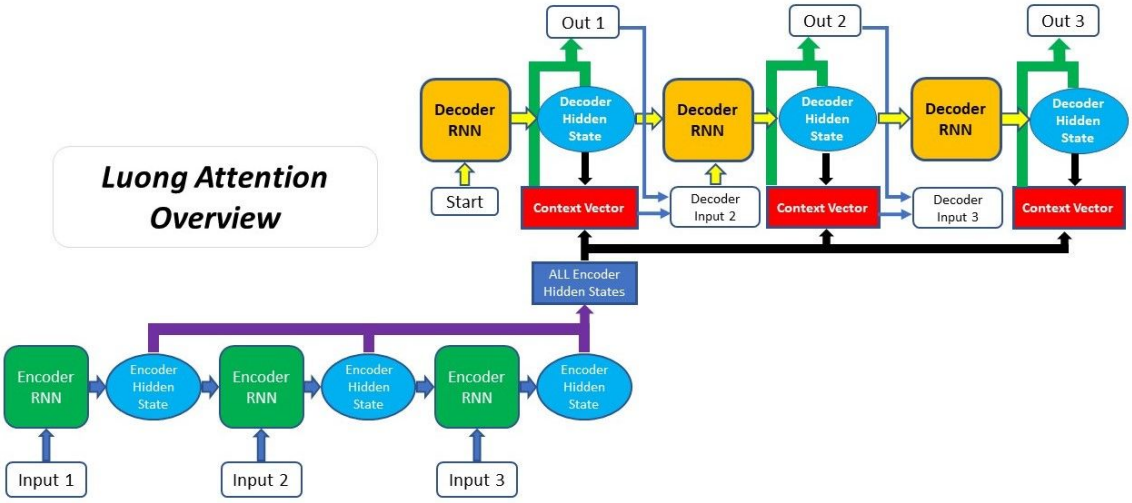


Fig. 2.22. Luong attention in sequence to sequence model [33].

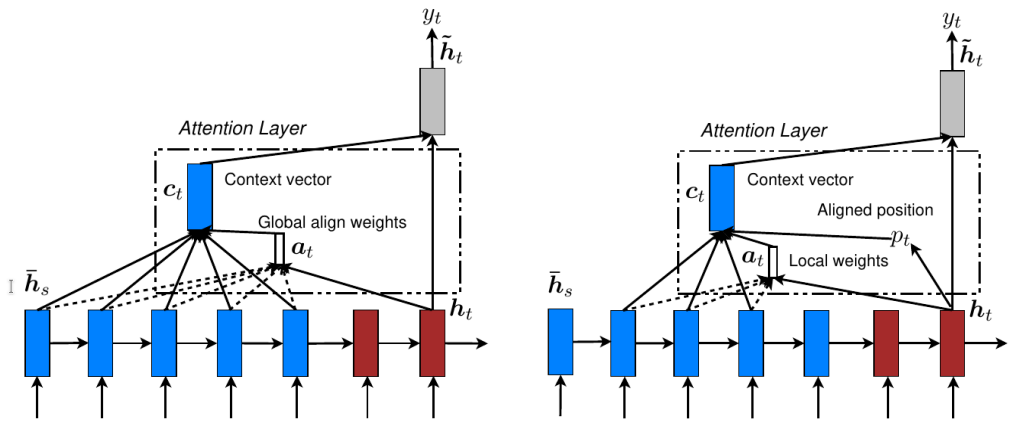
They also differ in the score function. They use three different types: defined-dot, general and concat [80]. However, only mention the **dot product attention** [26] for the purpose of this work.

In this case it is assumed that the query has the same dimensions as the keys $q,\ k_i \in \mathbb{R}^d$ for all i. The scores are computed by the dot product between the key and a key.

$$\alpha(q,k) = \frac{\langle q,k \rangle}{\sqrt{d}} \tag{2.27}$$

The scale by $\frac{1}{\sqrt{d}}$ is to avoid extremely small gradients for large values of d. It is explained in [27] how the dot product can grow in magnitude respect d, generating problematic gradients by the softmax function.

In this paper it is also explained another classification for attention. A **global** approach which always attends to all source words and a **local** one that only looks at a subset of words each time.



(a) Global attention (b) Local attention

Fig. 2.23. Differences between global and local attention [64].

2.7.2. Attention vs. Self-Attention

Attention mechanisms have become an essential component into NLP models, since they allow to model dependencies without regard to their distance in the input or output sequences. Different complex networks have been implemented as in [63], which it is used a deep LSTM with 8 encoder and 8 decoder layers using attention connections from the decoder network to the encoder.

Another example in [83] employs richer structural distributions within deep networks. This architecture corresponds with simple extensions of the basic attention procedure, such as attending to partial segmentations or to subtrees.

As it can be seen, attention is applied between the input and the output. The intuitive idea is to relate the source words with the target words.

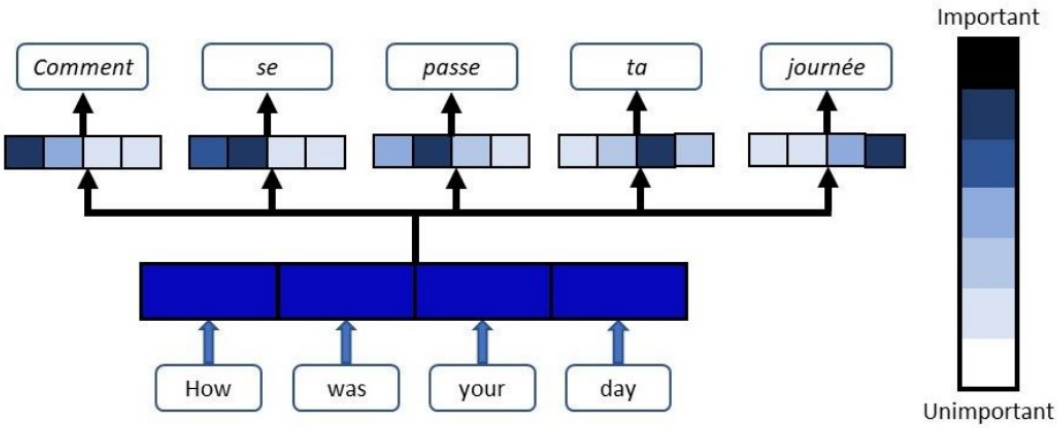


Fig. 2.24. Weights in attention [80].

From this idea comes **self-attention**. Instead of applying attention between the input and the output, this mechanism is only employed within the input elements. This concept is one of the main parts in this work, and it will be explained in more detail in section 3.

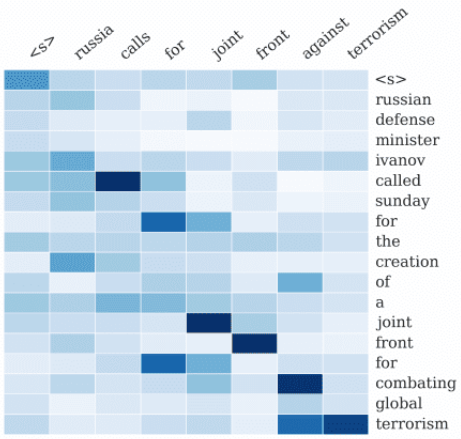


Fig. 2.25. Weights in self-attention [84].

Self-attention has been successfully implemented in muliple tasks such as summarization [85], [84], reading comprehension [86], embedding sentences [87] or natural language inference [88].

2.8. Competitive Models

Many efforts have continued to improve previous networks or strategies, pushing the performance to the limits. Modern networks were usually based on combination of advanced RNNs with atention mechanisms. It is also common the use of encoder-decoder arhitectures [89] or convolution methods.

End-to-end memory networks [90] consists in a recurrent attention model over a potential external memory. Following the architecture of a memory network [91] and trained end-to-end, requiring less supervision during training, the network is more generally applicable in realistic settings. It can be seen as an extension of the attention network used in [33], where multiple computational steps are performed per output symbol. It performs well on synthetic question answering [92] and language modelling.

The common goal is reducing sequential computation and its difficulty to train deep networks. **Neural GPUs** [93] was presented to address this problem. This network is based on a type of convolutional gated recurrent unit. It is highly parallel, making the model easier to train and efficient to run.

However, it did not get good results in machine translation. **Extended Neural GPU** [94] is an improved version which got better results over attention in algorithmic tasks, image processing, and in generative modelling. Instead of focusing in a single part of a memory, it is operated on all of it in parallel in a uniform way. This mechanism is called *active memory*. However, it did not get better result than attention for most natural language processing tasks.

One relevant novel network was **ByteNet** [95]. It is a one-dimensional convolutional neural network with a encoder-decoder structure. Both elements are interconnected by stacking the decoder on top of the encoder. ByteNet obtained state-of-the-art results on character-level language modelling and character-to-character machine translation on the English-to-German WMT translation task.

Another convolution network to mention is **ConvS2S** [96]. Its architecture is based entirely on convolutional neural networks to fully parallelized during training and exploit moder hardware (GPUs or TPUs). Optimization is also easy due to the number of non-linearities is fixed and independent of the input length. It is also used gated linear units to ease gradient propagation and attention module in each decoder. ConvS2S got also great results in English-to-German WMT translation task, in addition to English-to-French WMT.

3. SELF-ATTENTION

It is an attention mechanism relating different positions of a single sequence in order to compute a representation of this one. The goal is mapping sets to sets, regardless of the order in the sequence.

Self-atention was the approach to solve the problems given by the sequence-to-sequence models as it is explained at the presentation of attention in section 2.7. It is the essential component in the transformer and the new state-of-the-art strategy in NLP models.

3.1. Word Embeddings

If we have a sentence with length n, each word is usually represented with a word vector $x_i \in \mathbb{R}^d, i \in \{1, \ldots, n\}$, where d is the dimension of the vector. This parameter depends on the size of embedding we are using.

$$\begin{array}{ccccccccc} \text{The} & & \text{orange} & & \text{is} & & \text{very} & & \text{healthy} \\ x_1 & & x_2 & & x_3 & & x_4 & & x_5 \end{array} \tag{3.1}$$

These vectors do not provide context information, i.e. the relationship with the rest of the elements in the sequence. For example, x_2 does not determine if *orange* refers to the color or to the fruit, so it is not clear if it exists a clear relationship with *healthy*.

Then it is needed an output vector, formed by attention weigths, $y_i \in \mathbb{R}^n, i \in \{1, \ldots, n\}$, where n is the number of elements in the sequence. For all $i, j \in \{1, \ldots, n\}$.

$$y_i = \sum_j^n x_j w_{ij} \tag{3.2}$$

The question is how to get these weigths. In order to illustrate the process, it is going to be used the previous sentence and get the weights associated to the new embedding vector for *orange* (y_2).

Remember that n is the number of words in the sentence and d the dimension of the word vectors. The following explanation is based in [97].

3.2. Weighted Averages of Words

The traditional idea was based in providing larger weigths to similar words making use of the **dot product**. Two similar vectors are going to get a bigget dot product than two very

distant [98]. Applying the dot product and softmax respect x_2:

$$z_1 = x_1^T x_2 \quad \dots \quad z_n = x_n^T x_2 \tag{3.3}$$

$$\begin{bmatrix} w_1 & \dots & w_n \end{bmatrix} = \text{softmax}\begin{pmatrix} z_1 & \dots & z_n \end{pmatrix} \tag{3.4}$$

$$y_2 = \sum_{j=1}^{n} x_j w_j \tag{3.5}$$

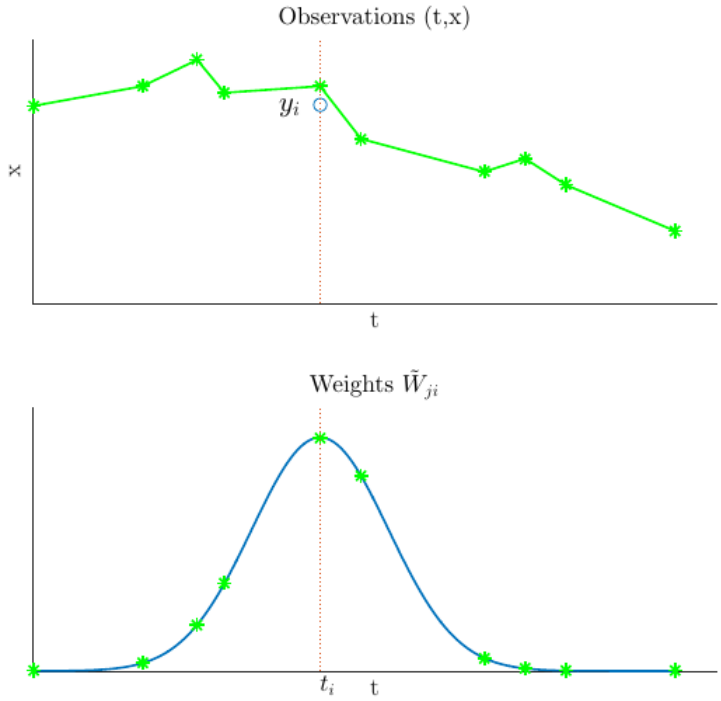


Fig. 3.1. Weighted Averages [97].

Intuitively, it is a method which could obatin good results. Nevertheless, several times correlated words are not necessary close. Therefore it is not an accurate and general method to extract the information between one element and the rest of the sequence.

3.3. Weights in Self-Attention

It is necessary to introduce three vectors: **query**, **keys** and **values**. It is also included three new parameters: W_Q, W_K, W_V, which are going to be trained. For all $i \in \{1, \dots, n\}$.

$$\begin{aligned} \text{Query} &\rightarrow q_i = W^Q x_i \\ \text{Keys} &\rightarrow k_i = W^K x_i \\ \text{Values} &\rightarrow v_i = W^V x_i \end{aligned} \tag{3.6}$$

Instead of using directly the inputs x_i, it is employed the **query** q_i and **keys** k_i to calculate

the weights.

$$z_1 = \frac{k_1^T q_2}{\sqrt{d}} \quad \dots \quad z_n = \frac{k_n^T q_2}{\sqrt{d}} \tag{3.7}$$

$$\begin{bmatrix} w_1 & \dots & w_n \end{bmatrix} = \text{softmax}\begin{pmatrix} z_1 & \dots & z_n \end{pmatrix} \tag{3.8}$$

To calculate the final embedding vector it is used the **values** v_i instead of the inputs x_i.

$$y_2 = \sum_{j=1}^{n} v_j w_j \tag{3.9}$$

Intuitively,the query attends the word for which we are going to get the attention weights. Keys corresponds with the rest of the elements in the dot product operation. Values are used in the final combination with the weights. Thanks to learnable matrixes, these combinations are optimized to find combinations and correlations in the own sentence.

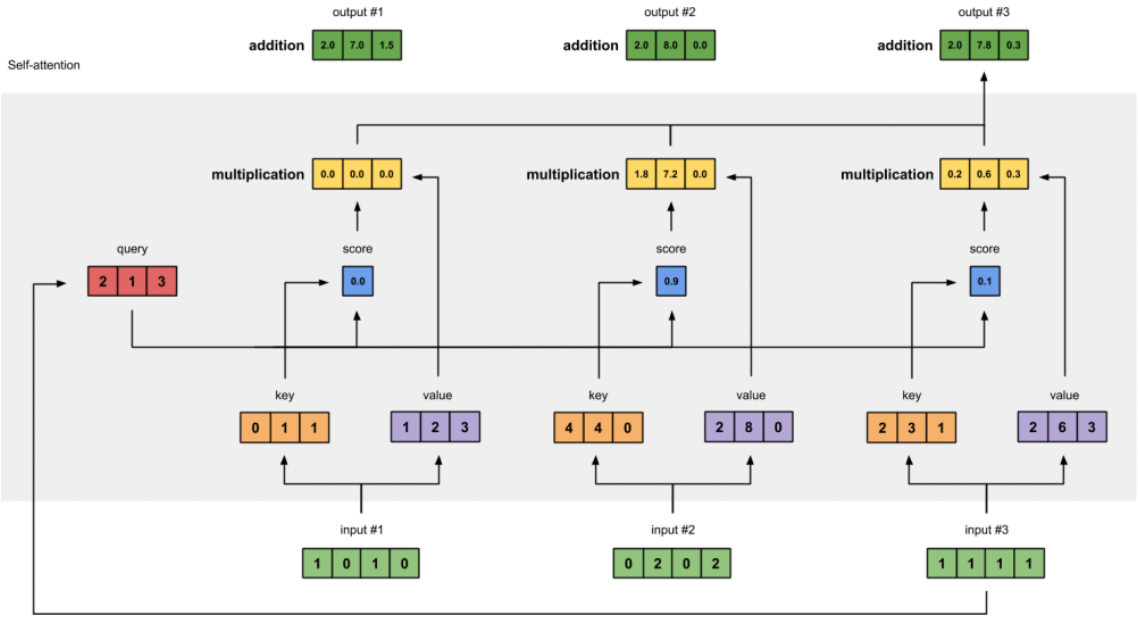


Fig. 3.2. Self-attention process [99].

3.3.1. Matrix Notation

To represent the whole process in matrix notation. Having a matrix **input**:

$$\text{Input} \quad \rightarrow \quad X = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix} \tag{3.10}$$

Calculate **query**, **keys** and **values**:

$$\begin{aligned} Q &= W^Q X = \begin{bmatrix} q_1 & \dots & q_n \end{bmatrix} \\ K &= W^K X = \begin{bmatrix} k_1 & \dots & k_n \end{bmatrix} \\ V &= W^V X = \begin{bmatrix} v_1 & \dots & v_n \end{bmatrix} \end{aligned} \tag{3.11}$$

Calculate the **weights**:

$$Z = \frac{K^T Q}{\sqrt{d}} \tag{3.12}$$

$$W = \text{softmax}(Z) \tag{3.13}$$

Finally, we get the **output** matrix:

$$\text{Output} \quad \rightarrow \quad Y = VW = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix} \tag{3.14}$$

At the end, we are looking for a rectangular matrix which relates every word in the sentence, as it is shown in figure 2.25. To get some better intuition we can use exBERT tool [100]. More details about how to use this tool will be given in section 5.2.3. At this point, it is enough to see how the connections between elements are plotted according to the attention weight, through line's thickness.

Fig. 3.3. Attention weights visualization using exBERT.

4. THE TRANSFORMER

In 2017 was presented the Transformer [27]. It is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNNs or convolution. The Transformer reached a new state of the art in translation quality, the original goal for the model.

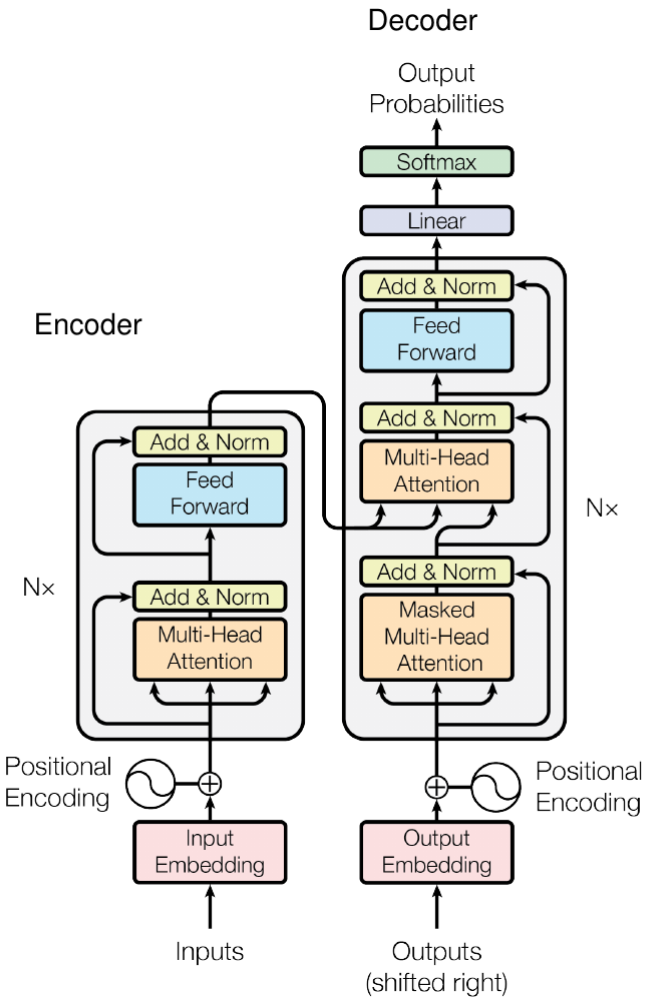


Fig. 4.1. Transformer's architecture [27].

In order to understand how the Transformer works, it is going to be explained the main parts of it. The code used in the notebook to implement the transformer is original from [101], which implements the model step by step as it is discussed in the original paper [27]. However, in the notebook we are only going to show the coding parts which give some intuition about the model and how the data is processed. To see more details and examples about how to tun the transformer it is recommended to check [101].

4.1. Model Architecture

The model is based in an encoder-decoder structure, as other competitive neural sequence transduction models. It receives as input a complete sentence and generates the probabilities for the next word as output. Usually it is selected the higher one and feed it into the decoder until it is selected the token which indicates the end of sequence.

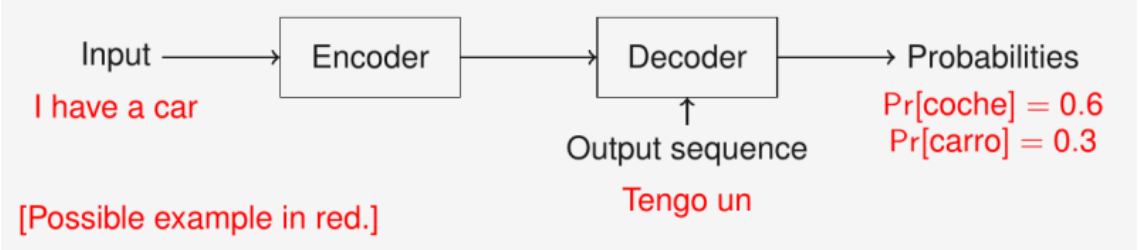


Fig. 4.2. Transformer's diagram [97].

The Transformer generates one symbol at a time, consuming the previously generated symbols as additional input, so it is an example of auto-regressive model [102].

4.2. Input Embedding

Before going into the encoder and decoder stacks, mention how the sentences are preprocessed. The word vectors used as input in the encoder and the decoder are formed by the sum of other two vectors: **word embedding** and **positional encoding**.

Let's define some variables in order to understand how data is managed by the transformer:

- Number of sentences in a batch: m.
- Number of words per sentence: n.
- Embedding dimension: d_{model}.
- Size of vocabulary: $vocab$.

4.2.1. Word Embedding

It is used learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model}. It is shared the same weight matrix between the two embbeding layers [103]. Also mention those weights are multiplied by $\sqrt{d_{model}}$.

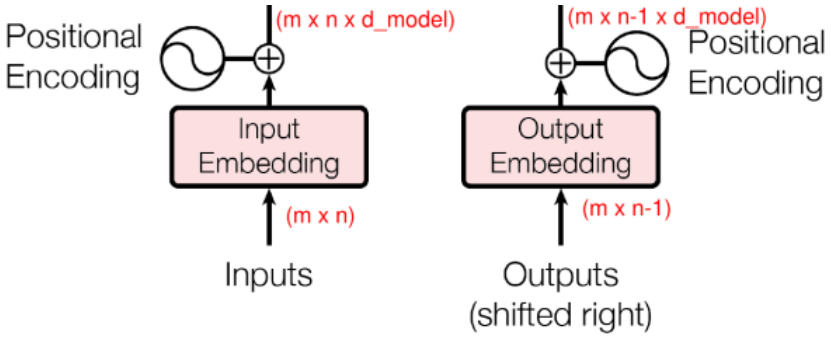


Fig. 4.3. Transformer's input embedding [27].

4.2.2. Positional Encoding

The attention blocks only map sets to sets, due to that, there is not any information about the position of the word in the sentence. Then it is injected some information about the relative or absolute position. In the original paper [27], it is used sine and cosine functions of different frequencies:

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{4.1}$$

The output is the addition of both embeddings.

$$x_i = \underbrace{E_i}_{\text{Word Embedding}} + \underbrace{P_i}_{\text{Positional Encoding}} \tag{4.2}$$

4.3. Encoder

The encoder stacks N encoder blocks which map sets to sets, what means the output size is the same than input: (m x n x d_{model}). In the original implementation, they employ $N = 6$ encoders.

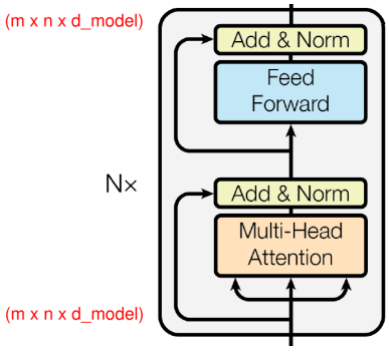


Fig. 4.4. Encoder's architecture [27].

Each block is formed by two sub-layers: **Multi-Head Attention** and **Feed Forward**. Both are followed by an **Add & Norm** layer.

4.3.1. Multi-Head Attention

The Transformer is differentiated for using exclusively self-attention. This model employs the **dot product attention** (equation 2.27). In [27] they called it **Scaled Dot-Product Attention**, due to the difference of using the scaled factor $\frac{1}{\sqrt{d}}$ or not.

Scaled Dot-Product Attention

It is much fast and more space-efficient in practice than additive attention (equation 2.26), since it can be implemented using highly optimized matrix multiplication code.

We have queries Q and keys K of dimension d_k and values V of dimension d_v.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{4.3}$$

As it is explained in Luong attention (section 2.7.1), for large values of d_k the softmax function could have extremely small gradients. For this reason, the dot product is scaled by $\frac{1}{\sqrt{d_k}}$.

In spite of using this type for the original implementation, it is important to consider that additive attention outperforms dot-product attention attention in case the scaling factor is not applied [104].

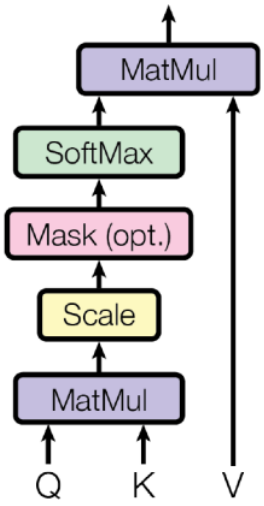


Fig. 4.5. Scaled Dot-Product Attention diagram [27].

Remembering that m is the number of sentences in a batch, n the number of words per sentence and d_{model} the embedding dimension, we are going to introduce two new variables to explain multi-head attention:

- Attention blocks: h.
- Dimension attention layers: $d_k = \frac{d_{model}}{h}$.

Multi-Head Attention is based on performing attention in parallel. There are h different blocks, with identical structure but different parameterers: W^Q, W^K and W^V. The output vectors from different heads are concatenated for each word using the parameter W^O.

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{4.4}$$

In this case we have $W_i^Q \in \mathbb{R}^{d_{model} \text{ x } d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \text{ x } d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \text{ x } d_v}$ and $W^O \in \mathbb{R}^{hd_v \text{ x } d_{model}}$.

The idea is to get information from different representation subspaces at different positions. To get an intuition, we could see it as one head is understanding the sentence gramatically, another one from the point of spelling, etc.

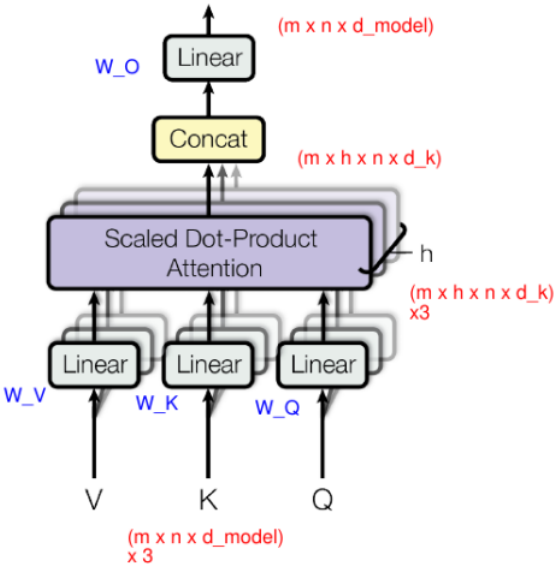


Fig. 4.6. Multi-Head Attention diagram [27].

The tensor is divided into h heads and then concatenated to recover its original shape. In [27] and [101] it is used $h = 8$ and they assume $d_v = d_k$.

4.3.2. Feed-Forward

Each layer contains a fully connected feed-forward network. It is used different parameters between blocks.

$$\text{FFN}(x) = \max(0,\ xW_1 + b_1)W_2 + b_2 \tag{4.5}$$

In [27] and [101] the input and output have dimensionality $d_{model} = 512$ and the inner-layer $d_{ff} = 2048$.

4.3.3. Add & Norm

This layer corresponds to a residual connection [105] followed by layer normalization [106] in both sub-layers. The goal of a residual connection is easing the training, with layers learning functions with reference to the layer inputs, instead of learning unreferenced ones. The layer normalization is a normalization which is applied per feature.

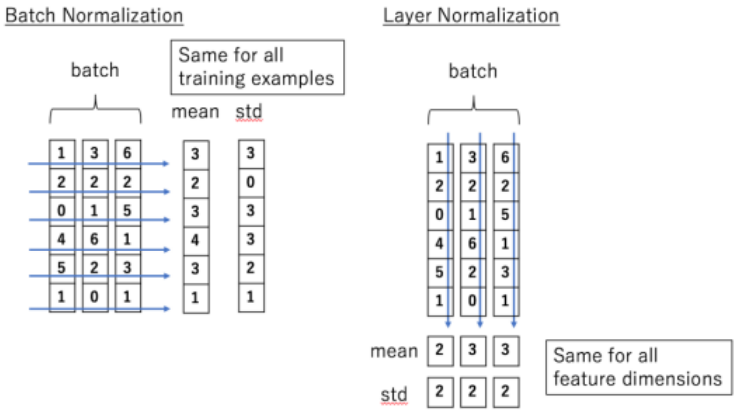


Fig. 4.7. Layer normalization [50].

Then, the output of each sub-layer is:

$$\text{LayerNorm}(x + \text{Sublayer}(x)) \tag{4.6}$$

where Sublayer(x) is the function implemented by the corresponding sub-layer.

In order to reduce overfitting, it is applied dropout to the output of each sub-layer, before it is added to the sub-layer input and normalized [107].

4.4. Decoder

It is also composed of a stack of N decoder blocks. Using the output from the encoder its function is also mapping sets to sets, maintaning the shape: number of vectors m and length n. However, the dimensionality in the number of words differs one position from the encoder, being the size: (m x n − 1 x d_{model}). In the original implementation, they employ $N = 6$ decoders.

The architecture is almost equal to the encoder. Appart from the dimension, there are other two differences:

- The mask for the first multi-attention.
- The encoder's output used as input in the second multi-attention.

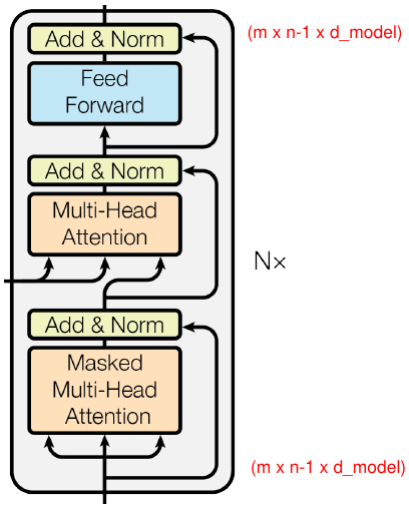


Fig. 4.8. Decoder's architecture [27].

4.4.1. Masked Multi-Head Attention

This mechanism prevents attending to subsequent positions and ensures that the predictions only depend on known outputs previous to the current one. In figure 4.9 from [50], we can get an intuition about the process.

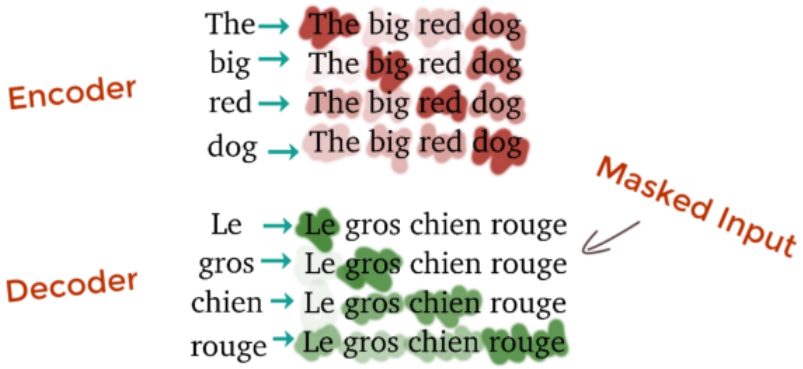


Fig. 4.9. Self-attention intuition in encoder and decoder [50].

4.4.2. Encoder-Decoder Self-Attention

The second sublayer in the decoder is a multi-head attention, without mask in this case, but using the encoder's output as input in the block. In table 4.1 it is summarized.

Query	Q^D	From decoder's input embedding
Keys	K^E	From encoder's output
Values	V^E	From encoder's output

Table 4.1. INPUTS SECOND MULTI-HEAD ATTENTION BLOCK IN TRANSFORMER'S DECODER.

4.5. Generator

The final block is a combination of a feed-forward network and a softmax in order to get probabilities for next word.

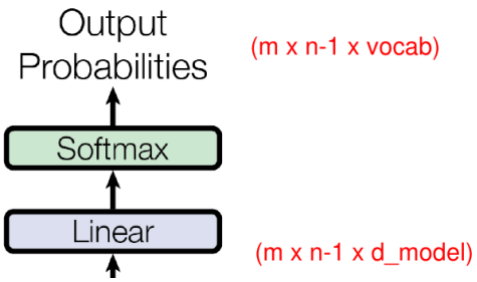


Fig. 4.10. Generator's architecture [27].

4.6. Transformer improvements due to self-attention

The transformer involved a critical change and improvement in current NLP models due to the given use of self-attention.

As it is explained in [27], self-attention connects all positions with a constant number of sequentially executed operations, whereas a RNN needs $O(n)$ sequential operations. Respect computational complexity, self-atention layers are faster when sequence length n is smaller han representation dimension d, as in most of the cases [63], [108].

Transformers has also its advantages over convolutional networks. Connect all pairs of input and output positions requires a stack of convolutional layers $O(n/k)$ or $O(\log_k(n))$ applying dilated convolutions [95]. The last one increases the length of the longest paths between any two positions, deriving into the mentioned problem about learning long-term dependencies [38]. Best case is separable convolutions [70], with a complexity of $O(k \cdot n \cdot d + n \cdot d^2)$. However, the combination of self-attention layer and a point-wise feed-forward layer as in the transformer results in the same complexity and a more interpretable model.

5. BERT

Bidirectional Encoder Representations from Transformers (BERT), released in 2019 in [9], obtaining new state-of-the-art results on eleven NLP tasks.

BERT was designed to pretrain deep bidirecional representations from unlabeled text by using self-attention to get the left and right context. This approach overcomes current pre-training techniques, where they use unidirectional language models to learn general language representations. Two examples which were competitive models before BERT are ELMo [109] and Generative Pre-Trained Transformer (GPT) [110].

This pretrained model can be fine-tuned adding one output layer and used for a wide range of NLP tasks.

The original model developed in TensorFlow is in github.com/google-research/bert, but it is going to be used the libray of Hugging Face [17] to show the model.

5.1. Model Architecture

It is a multi-layer bidirectional Transformer encoder, based on [27] and implemented as [101]. The different hyperparameters and the original notation are:

- Number of layers: L, number of encoder blocks which compose the whole stack. Previously defined as N in the Encoder part.
- Hidden size: H, embedding dimension of the model. Previously defined as d_{model}.
- Self-attention heads: A, number of heads in self-attention blocks. Previously defined as h.

There are two model sizes defined. In both cases the inner layer for the last feed-forward block is defined as $4H$. In table 5.1 we can check the number of parameters in each case.

Parameters	$\text{BERT}_{\text{BASE}}$	$\text{BERT}_{\text{LARGE}}$
Encoder Blocks (L)	12	24
Hidden size (H)	768	1024
Self-attention heads (A)	12	16
Inner layer feed-forward (4H)	3072	4096
Total parameters	110M	340M

Table 5.1. NUMBER OF PARAMETERS COMPARATION BETWEEN $\text{BERT}_{\text{BASE}}$ AND $\text{BERT}_{\text{LARGE}}$.

BERT has been already trained, so we can instantiate one of the base model classes of the HuggingFace library from a pretrained model. To load the required model is used the class BertModel. HuggingFace has also the the possibility to differentiate between cased and uncased models, a feature which was not originally implemented.

5.1.1. Input-Output Representations

In order to handle a variety of tasks, the input can be a single sentence or a pair of sentences in one token sequence, which is certainly the name given to the input. The input representation of a word is the addition of three different embeddings:

- **Token Embedding**: normal word embedding. It is used WordPiece [111] with a 30.000 token vocabulary.
- **Segment Embedding**: to differentiate the sentence the word belongs to. It is used the special token [SEP] to separate both sentences.
- **Position Embedding**: to indicate the position in the whole sequence.

The first token is always a special classification token [CLS], whose final hidden state layer is used in classification tasks.

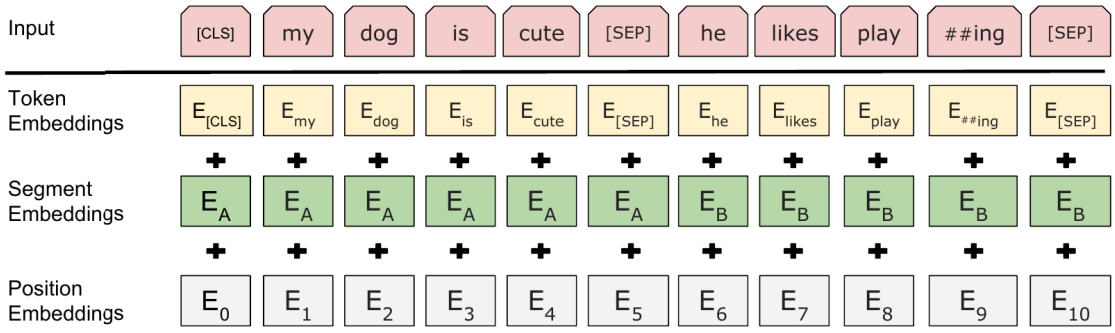


Fig. 5.1. Input Embedding in BERT [9].

As output we get the corresponding sequence embedding, as it is explained in the encoder, and NSP for classification. These values are going to be explained better in Pre-Training section 5.2.

5.2. Pre-Training

This step is in charge of the model to understand the language and its context. It is used two unsupervised tasks instead of the traditional left-to-right or right-to-left.

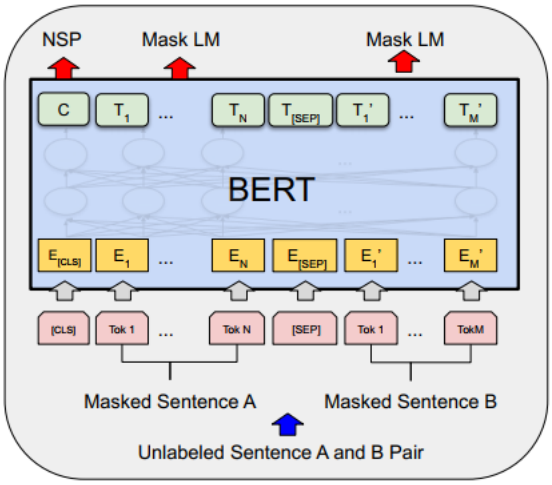


Fig. 5.2. Pre-training process in BERT [9].

They used as pre-training data the BooksCorpus [112] and English Wikipedia (2,500M words). It is recommended in [9] to use a document-level corpus rather than a shuffled sentence-level corpus in order to extract long contiguous sequences.

Training of $\text{BERT}_{\text{BASE}}$ was performed on 4 Cloud TPUs (16 TPU chips total). While training of $\text{BERT}_{\text{LARGE}}$ employed 16 Cloud TPUs (64 TPU chips total). Each pre-training took 4 days to complete [9].

5.2.1. Masked Language Model (MLM)

Some tokens are masked at random and then predicted. The final T_i vectors have the sime size and are generated simultaneously. They are fed into an output softmax over the vocabulary (30k) to get a distribution and compare with cross entropy loss in order to train the model.

However, as the output have all the worlds, included the ones were not masked, it is only considered the ones were masked and ignores the others to calculate the cross entropy loss. This ensures more focus into predict the masked values and increases the context awareness [113] (figure 5.3).

5.2.2. Next Sentence Prediction (NSP)

Many important tasks as question and answering are based on understanding relationship between two sentences, which is not captured by language modeling [9]. For that reason, while the model is being trained in MLM), is also trained in next NSP.

For this purpose it is used the special token [CLS] in the input and C in the output to indicate if *Sentence B* could follow *Sentence A*.

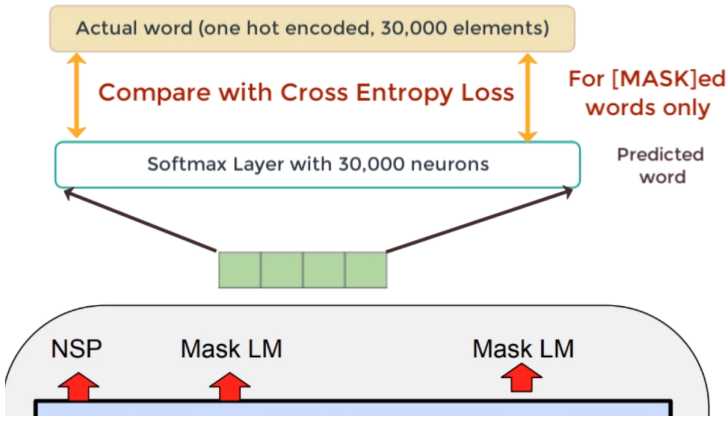


Fig. 5.3. Prediction of masked word in BERT [113].

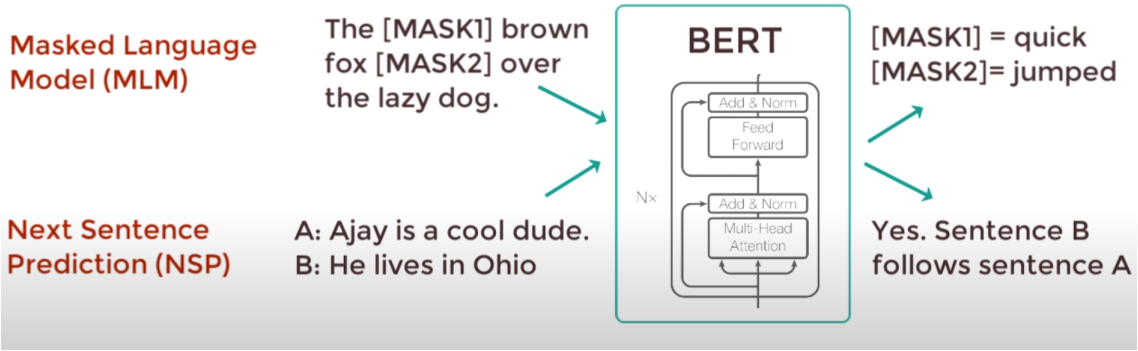


Fig. 5.4. MLM and NSP in pre-training BERT [113].

5.2.3. Self-Attention Visualization

As it has been mentioned, the model contains L blocks and A attention heads. Many models which come from BERT acquired the encoder to perform self-attention. Self-attention operations are not straightforward, even further if we have different blocks and many heads into them.

To get an intuition about BERT (and other similar models) respect the attention weights between the elements, we can use **exBERT** [100]. Following next steps:

1. First we select a model and introduce the input sentence.
2. We can hide special tokens and choose the percentage of attention shown.
3. We select an encoder block (as *Layer*).
4. Last, the attention heads we want to visualize.

One example of the application is shown in figure 5.5.

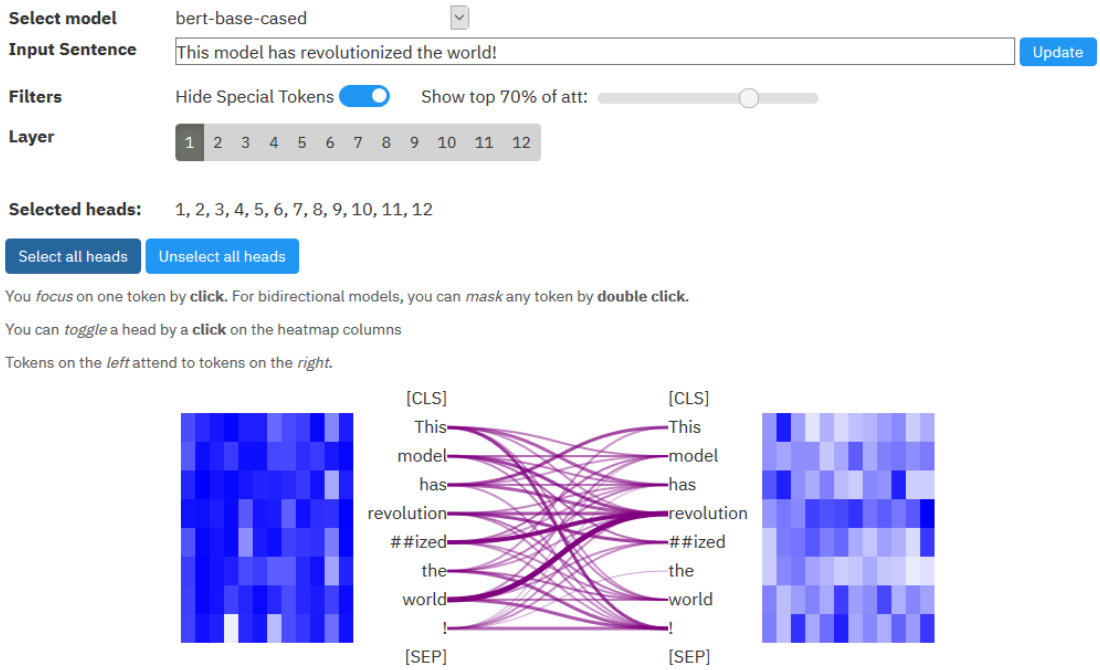


Fig. 5.5. Example of exBERT visualization.

5.3. Fine-Tuning

Thanks to self-attention, which is in charge of applying bidirectional cross attention between the two sentences, is straightforward to model many downstream tasks. In addition to, the two sentences as input in pretraining are equivalent to different trainings in NLP as sentence pairs in paraphrasing, hypothesis-premise pairs, question-passage pairs, etc.

The output token representations are fed into an output layer and the C is fed into an output layer for classification. Only new parameters are learned from sratch, while the rest are slightly fine-tuned (figure 5.7).

That is the reason fine-tune is relatively inexpensive in computation cost compared with pre-training. In the original paper, authors claim that all of the results can be replicated in at most 1 hour on a single Cloud TPU, or a few hours on a GPU, starting from the exact same pre-trained model.

The original experiments in the paper [9] correspond with text classification on GLUE [114] and question answering on SQuAD [115]. It is also evaluated on SWAG [116], where given a sentence, the task is to choose the most plausible continuation among four choices.

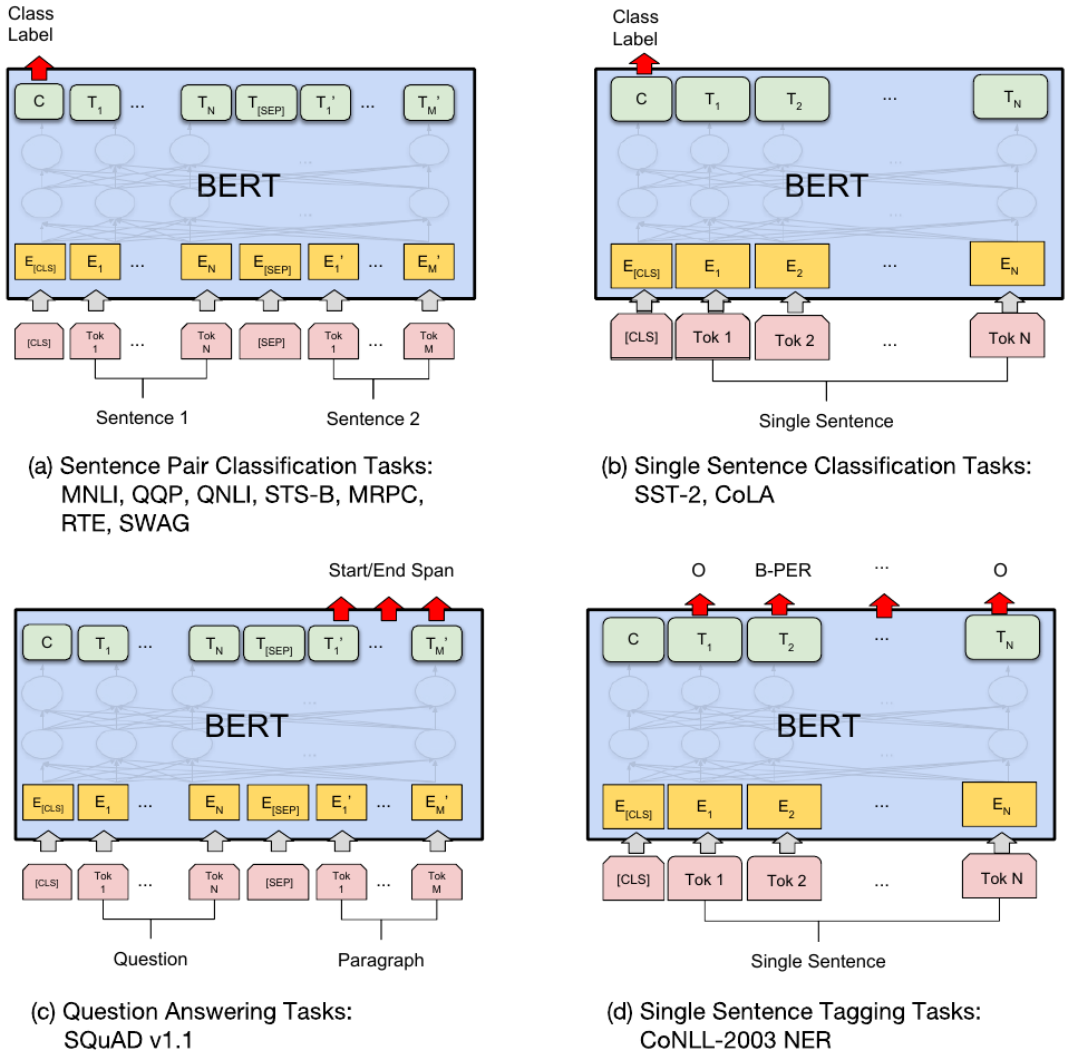


Fig. 5.6. Different NLP tasks implemented by fine-tuning BERT [9].

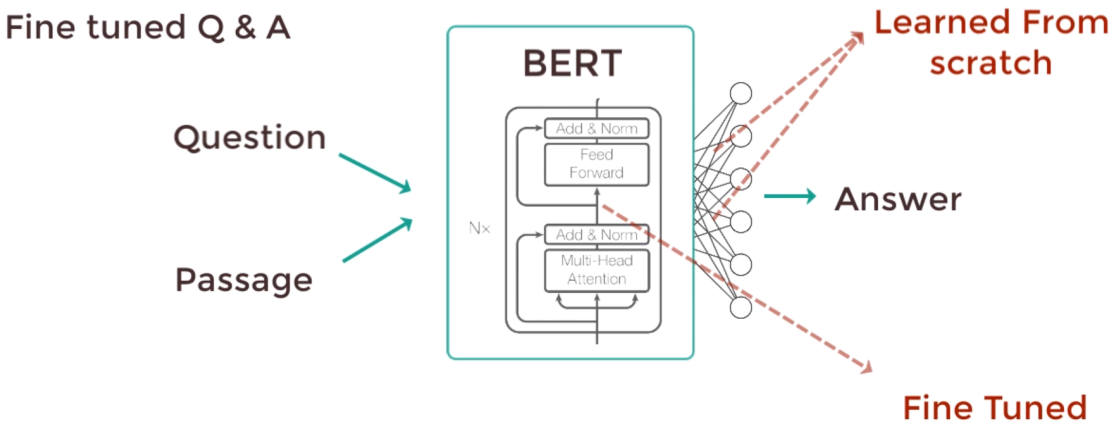


Fig. 5.7. Output layer added in fine-tuning [113].

6. SENTIMENT ANALYSIS

A tradiditonal task in NLP is sentiment analysis. The target is to determine if the sentence has a *positive* or *negative* sentiment, corresponding with a binary classification problem. As it is normal, these two labels depend on the context of our text. In spite of being one of the most simple applications, it is really useful and currently demanded in many applications, for example in reviews of films and series.

In addition to be a standard assignment, it compounds one of the task of General Language Understanding Evaluation (GLUE) [114], a group of nine classification tasks on sentences or pairs of sentences, which is commonly used to evaluate state-of-the-art models. Sentiment analysis corresponds with Stanford Sentiment Treebank (SST-2) [117].

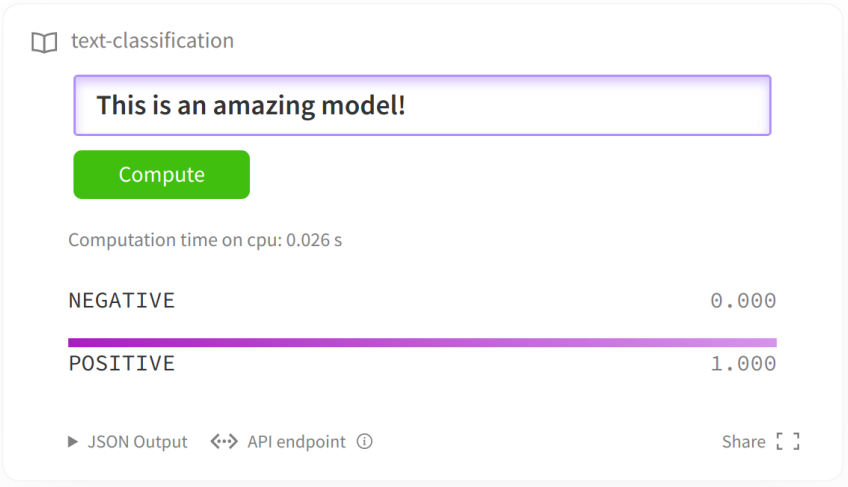


Fig. 6.1. Example of sentiment analysis in one of the Hugging Face's models [118].

For this task, two tutorials [119], [120], have been used as guide and support to the development of the application.

6.1. Setup

In spite of being much faster than pre-training, fine-tune a BERT model for any class could be slow for a CPU. Due to that, it is going to be used a GPU provided by Google Colab. In this case it was assigned: **Tesla P100-PCIE-16GB**.

6.2. Load SST-2 Dataset

First of all we need to download the dataset from GLUE baseline [114] to the Colab instance's file system and unzip it. There are three different datasets: *train.csv*, *test.csv*,

dev.csv. As its own name indicates, they are used for training, testing and validate respectively, the studied model. As there is a GLUE leaderboard, the test dataset is not labeled. In order to study the performance of our model, we are going to divide the train dataset, which has enough samples, into train and test. Test set is going to be one third of the original train test.

To get a fast intuition on the dataset, we plotted some used sentences for the train split in figure 6.2. Mention the label 1 corresponds with positive sentiment, while label 0 with negative as it is shown in table 6.1.

Class	Sentiment
1	Positive
0	Negative

Table 6.1. LABELS IN SST-2 DATASET.

Some sentences of train split, which has a length of 45123

	sentence	label
3383	what with all the blanket statements and dime-store ruminations on vanity , the worries of the rich and sudden wisdom , the film becomes a sermon ...	0
40351	manages to squeeze by on angelina jolie 's surprising flair for self-deprecating comedy .	1
57620	with a completely predictable plot , you 'll swear that you 've seen it all before , even if you 've never come within a mile of the longest yard .	0
55502	are both excellent ,	1
52258	their struggle is simply too ludicrous and borderline insulting .	0

Fig. 6.2. Some samples of SST-2 training set.

Depending on the computation velocity, training process could be very slow. For a fast intuition on the model, we can remove some data, since only the training set has more than 45K samples. With *perc* we select the percentage of data we are going to use as it is shown in figure 6.3. Removed sample are selected randomly.

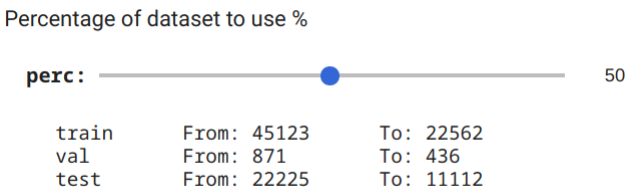


Fig. 6.3. Resize STT-2 dataset.

However, the result are provided using the whole dataset. In table 6.2 it is specified the number of samples per set.

It is important to know how many samples we have per class in our training dataset, in case we are dealing with an imbalanced classification problem. The number of samples is going to depend on how the original train datatset has been splitted to get the test set. In

Training	Validation	Test
45123	871	22225

Table 6.2. NUMBER OF SST-2 SAMPLES PER SET USED.

table 6.3 it is shown the samples classification for the used model. As it can be seen, the number of samples per class is balanced.

Total samples	Positive	Negative
45123	25114	20009

Table 6.3. NUMBER OF POSITIVE AND NEGATIVE SAMPLES IN SST-2 TRAINING SET.

6.3. Preprocessing SST-2 Data

Before input sequences into BERT, it is necessary to preprocess data samples. The steps to perform depend on the model, but mainly the guide to follow is:

- Tokenize sequences, converting text into integers according to embedding space used.
- Pad each sentence to the maximum length which is in your batch.
- Truncate each sentence to the maximum length the model can accept (if applicable).

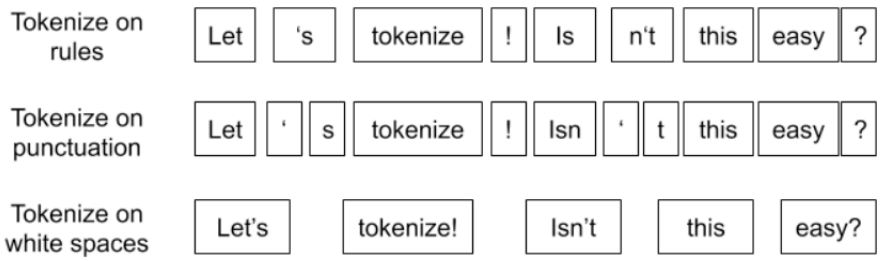


Fig. 6.4. Different types of tokenization [121].

The whole text must also be divided into sentences. Each one can select the more suitable approach depending on the origin of the text. In this case, the text is already divided into sentences.

BERT employs **WordPiece** [111] subword tokenization algorithm. Subword tokenization algorithms rely on the principle that frequently used words should not be split into smaller subwords, but rare words should be decomposed into meaningful subwords.

WordPiece is very similar to Byte-Pair Encoding (BPE) [122]. Every character presented in training data is included to learn the corresponding merge rules. In contrast to BPE that chooses the most frequent symbol pair, this algorithm maximizes the likelihood [123] of the training data when it is added to the vocabulary. This approach evaluates if it is worthy to merge two symbols, instead of separated.

Remember that BERT input follows next scheme:

[CLS] Sentence A [SEP] Sentence B [SEP]

or

[CLS] Sentence [SEP]

depending on the final task. For classification of one sentence, it corresponds with second option.

In order to preprocess according to the model, it is going to be used the BertTokenizer from Hugging Face, which ensures geting a tokenizer that corresponds to the desired model architecture. Besides that, it downloads the vocabulary used when pretraining this specific checkpoint.

It is necessary to pass to the tokenizer the name of the pre-trained model we are going to use. It is selected bert-base-uncased, a $\text{BERT}_{\text{BASE}}$ which does not distinguish between upper and lower case.

From tokenizer's instance it can be obtained information about the model. These values can be checked in table 6.4.

- Dimension of the vocabulary used
- Maximum allowed length of the model
- In which size is applied padding
- Special used tokens for separation token, padding token, classification token, mask token and other unkown tokens.

BERT works with sentences of same dimension. Due to that, it is required to pad (add padding tokens until a specified size) or truncate (separate a sentence if it is longer than the maximum model length). For this reason it is very useful to know which is the maximum length in our sentences, and limit the input size to this value. In this case, the maximum size was **66**.

In addition to, it is necessary to add the special tokens: [CLS] at the beginning of a sentence and [SEP] between them, altough in this task it will be only at the end of each one.

When a sentence is tokenized it is obtained the corresponding token IDs as and the attention mask. The mask indicates which tokens are useful and which ones are only padding,

Attribute	Value
vocab size	30522
model max length	512
padding side	Right
sep token	[SEP]
pad token	[PAD]
cls token	[CLS]
mask token	[MASK]
unk token	[UNK]

Table 6.4. PARAMETERS OF $\text{BERT}_{\text{BASE}}$.

a necessary requirement for passing an input to BERT. An example of the final preprocess result is shown in figure 6.5.

```
Original sentence:
one but two flagrantly fake thunderstorms

Token IDs:
[  101  2028  2021  2048  5210 17884  2135  8275  8505 19718  2015   102
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0]

After tokenization:
['[CLS]' 'one' 'but' 'two' 'flag' '##rant' '##ly' 'fake' 'thunder'
 '##storm' '##s' '[SEP]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]'
 '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]'
 '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]'
 '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]'
 '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]'
 '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]' '[PAD]'
 '[PAD]' '[PAD]' '[PAD]']
```

Fig. 6.5. Visualization of a tokenized sample.

Finally mention the generation of an iterator for our dataset using the torch DataLoader class. This will help save on memory during training and boost the training speed [124].

6.4. Generation of the Model

To fine-tune on GLUE, it is introduced classification layer weigths $W \in \mathbb{R}^{K\ x\ H}$, where K is the number of labels. Then it is computed a standard classification loss with C (explained in section 5.3) and W:

$$\log(\text{softmax}(CW^T)) \tag{6.1}$$

In sentiment analysis task, the final model is just a combination of BERT's encoder and a classifier to get the final probabilities.

Hugging Face provides BertForSequenceClassification to perform this task. However, in order to show the simplicity of fine-tune process, it is going to be created an own model.

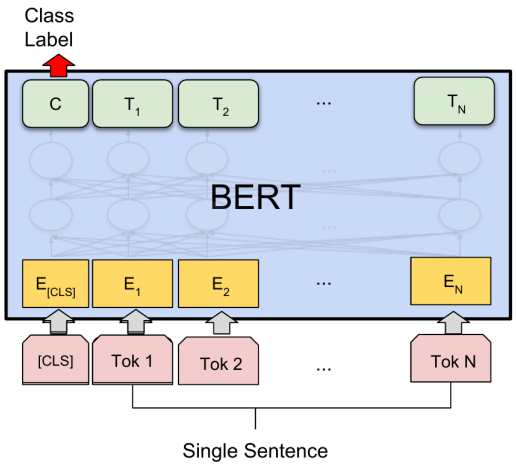


Fig. 6.6. Fine-tune BERT on sentiment analysis [9].

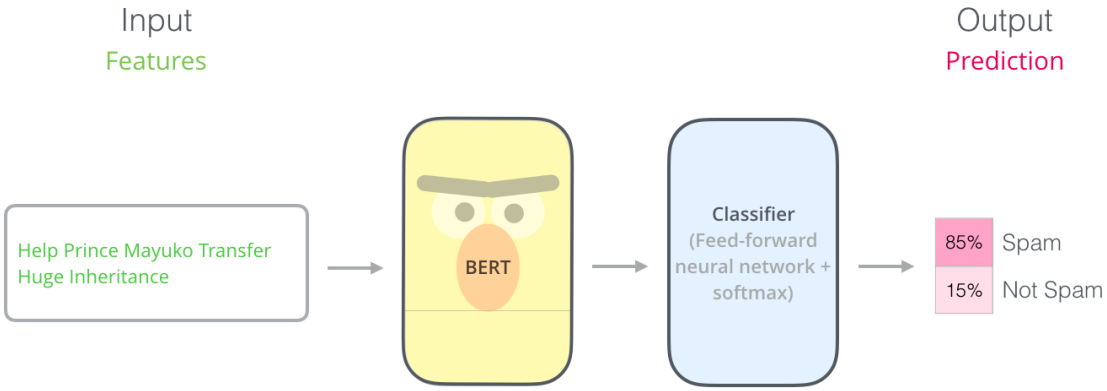


Fig. 6.7. Diagram of BERT Classifier [125].

At the end, this is the combination of pre-trained BertModel and a classifier, composed by a simple feed-forward neural network with only one hidden layer, plus a softmax to get the probabilities of each class.

Mention the own library launchs a warning when we load the base model. The message is telling us we are throwing away some weights and randomly initializing some other. That is because it is being removed the head used to pretrain the model on a masked language modeling objective and replacing it with a new head for which there are not pretrained weights. Then the library warns us we should fine-tune this model before using it for inference, which is exactly what it is done in section 6.5.

Respect classifier's parameters, it is selected the number of neurons in the hidden layer of the neural network and a dropout probability.

Finally, it is created an optimizer. In the original paper [9] they use an Adam optimizer [126]. The optimal hyperparameter values are task-specific, but they found a range of possible values to work well across all tasks. These ones are shown in table 6.5.

Parameter	Recommended values
Batch size	16, 32
Learning rate	5e-5, 3e-5 or 2e-5
Number of epochs	2, 3, 4

Table 6.5. RECOMMENDED HYPERPARAMETERS FOR FINE-TUNING BERT.

6.5. Fine-Tune on Sentiment Analysis

Before going into detail about how to fine-tune the model, a recap about binary classification problems, which consits in predicting a class between two possible ones. There are 4 possible combinations in this type of problems. These combinations are reflected in the **confusion matrix**, figure 6.8.

- **True Positive (TP)**: if both predicted and actual values are *Positive*.
- **False Positive (FP)**: if predicted value is *Positive* and actual one is *Negative*.
- **False Negative (FN)**: if predicted value is *Negative* and actual one is *Positive*.
- **True Negative (NP)**: if both predicted and actual values are *Negative*.

		Predicted	
		Positive	Negative
Real	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Fig. 6.8. Confusion matrix. Predicted vs. real values.

One method to measure the performance of the model is the **accuracy** [127]. This is defined as the coefficient between the number of correct predictions and the total number of predictions. Then, with previous classification possibilities defined we have:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.2}$$

In addition to the accuracy, it is used the cross-entropy loss [128]. Each sample has a known class label with a probability of 1.0, and 0.0 for the other label. The model can estimate the probability of an example belonging to each class label. Cross-entropy can then be used to calculate the difference between the two probability distributions.

To train the model, as in other architectures, we are going to iterate through batches. At the end of each epoch the model is evaluated respect the validation dataset. We use a fixed seed [129] for the random number generator to ensure the same result in two consecutive executions with same parameters.

It is declared a class called *BertTrainer* which is in charge of the training process. Selected parameters to train the model are shown in table 6.6.

Parameter	Value
Hidden size classifier	50
Dropout classifier	0.1
Batch size	32
Learning rate	5e-5
Epochs	3
Epsilon	1e-8

Table 6.6. SELECTED PARAMETERS FOR BERT CLASSIFIER.

```
Using GPU Tesla P100-PCIE-16GB.

======== Epoch 1 / 3 ========

Training: 1411 batches

 Batch   |  Train Loss   |  Elapsed
-----------------------------------
  20     |    0.607799   |  0:00:05
  40     |    0.418825   |  0:00:04
  60     |    0.398746   |  0:00:04
  80     |    0.361606   |  0:00:04
  100    |    0.315889   |  0:00:04
  120    |    0.301173   |  0:00:04
  140    |    0.294156   |  0:00:04
  160    |    0.232642   |  0:00:04
  180    |    0.284203   |  0:00:04
  200    |    0.254509   |  0:00:04
```

Fig. 6.9. Visualization of training process for BERT classifier.

Training results are shown in table 6.7. In addition to, we have plotted how both (training and validation) losses evolve during epochs in figure 6.10. Total training time was **27 minutes and 9 seconds**.

Epoch	Training Loss	Validation Loss	Accuracy	Training Time	Validation Time
1	0.224070	0.199047	0.916773	0:05:10	0:00:02
2	0.095247	0.199060	0.933036	0:05:09	0:00:02
3	0.043675	0.222848	0.935268	0:05:09	0:00:02

Table 6.7. BERT CLASSIFIER TRAINING RESULTS.

While the the training loss is going down with each epoch, the validation loss is increasing. This suggests that the model is being trained too long and it is over-fitting on the training data.

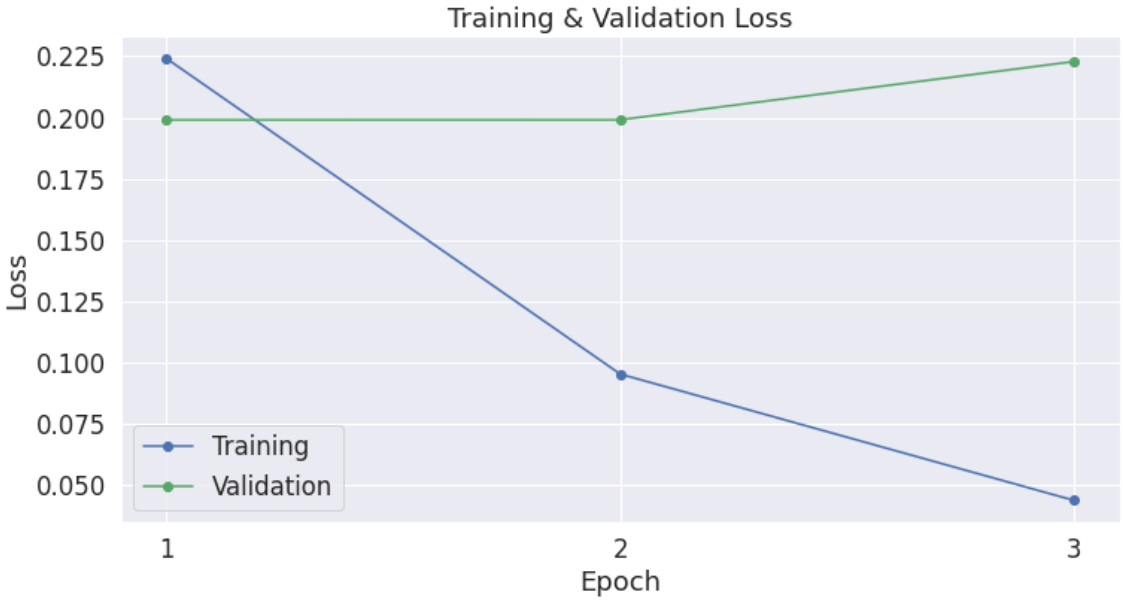


Fig. 6.10. Training vs. validation loss of BERT classifier.

6.6. Evaluation of the Model

Once the model is fine-tuned, it is evaluated respect the test dataset. It is needed to define the prediction function which is going to be almost equal to the evaluation function, but applying a **softmax** to return the probabilities per class. Results are presented in table 6.8.

Test Loss	Accuracy	Test Time
0.197639	0.949167	0:00:47

Table 6.8. BERT CLASSIFIER TEST RESULTS.

To get a fast intuition on the result, it is plotted some predicted sentences in figure 6.11. Remember that the label 1 corresponds with positive sentiment while label 0 with negative.

	sentence	label	real	predicted	probability
1891	the high-buffed gloss and high-octane jolts	1	positive	positive	0.999506
20706	the one-liners are snappy , the situations volatile and	1	positive	positive	0.999150
14030	of ideas for the inevitable future sequels (hey , do n't shoot the messenger)	0	negative	negative	0.991766
15419	it 's crafty , energetic and smart	1	positive	positive	0.999565
17873	make this surprisingly decent flick	1	positive	positive	0.999547

Fig. 6.11. Some predictions of BERT classifier on STT-2.

6.6.1. Variable Threshold

Sentiment classification task is even difficult for human. Therefore, define a threshold is going to provide a safer margin to classify one class [130].

For example, if it is defined that positive sentiment sample are the only ones where probability of being *positive* is bigger than 0.9 instead of 0.5, samples classified as positive are going to have surely positive sentiment. The main drawback here is all samples wich are positive and are classified as negative.

To get a better intuiton about the performance, we plotted the confusion matrix in figure 6.12.

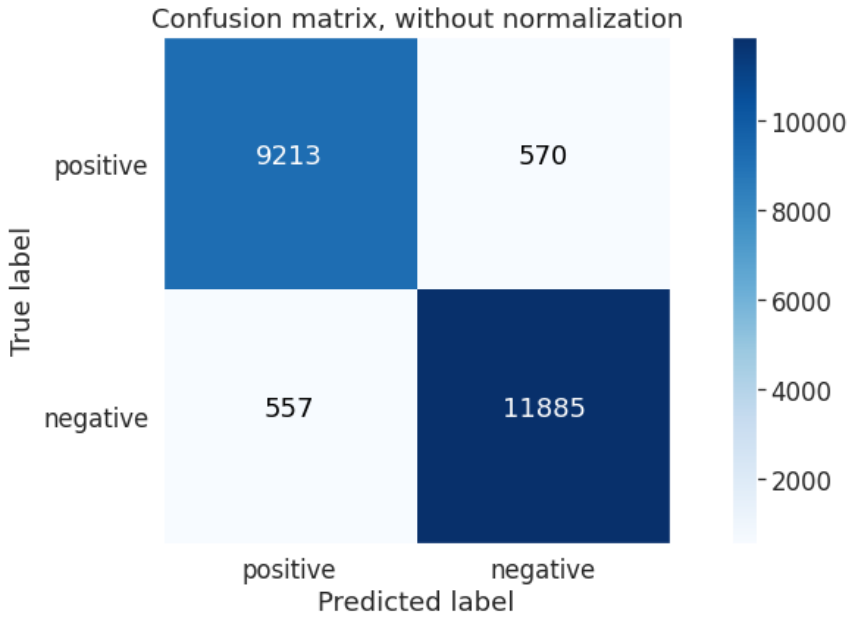


Fig. 6.12. Confusion matrix with 0.5 as threshold for BERT classifier.

A useful tool to determine an optimal threshold is the **Receiver Operating Characteristic curve (ROC)** [131] [132]. It is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different candidate threshold values between 0.0 and 1.0.

The **true positive rate** is calculated as the number of true positives divided by the sum of the number of true positives and the number of false negatives. It is also called **hit rate** and describes how good the model is at predicting the positive class when the actual outcome is positive. It is also known as **sensitivity** or **recall**.

$$TPR = \frac{TP}{TP + FN} \tag{6.3}$$

The **false positive rate** is calculated as the number of false positives divided by the sum of the number of false positives and the number of true negatives. It is also called the **false alarm rate** as it summarizes how often a positive class is predicted when the actual

outcome is negative.

$$FPR = \frac{FP}{FP + TN} \tag{6.4}$$

The complement of the FPR is the **specificity** and it is calculated as:

$$\text{Specificity} = 1 - FPR \tag{6.5}$$

The geometric mean or g-mean is a metric for imbalanced classification that, if optimized, will seek a balance between the sensitivity and the specificity [133].

$$\text{G-Mean} = \sqrt{\text{Sensitivity} \cdot \text{Specificity}} \tag{6.6}$$

One approach to determine the optimized threshold would be to test the model with each threshold and select the one with the largest G-Mean value. In this case the best one was **0.53** with a g-mean of **0.95** and AUC equals **0.99**. The ROC curve for the model is plotted in figure 6.13.

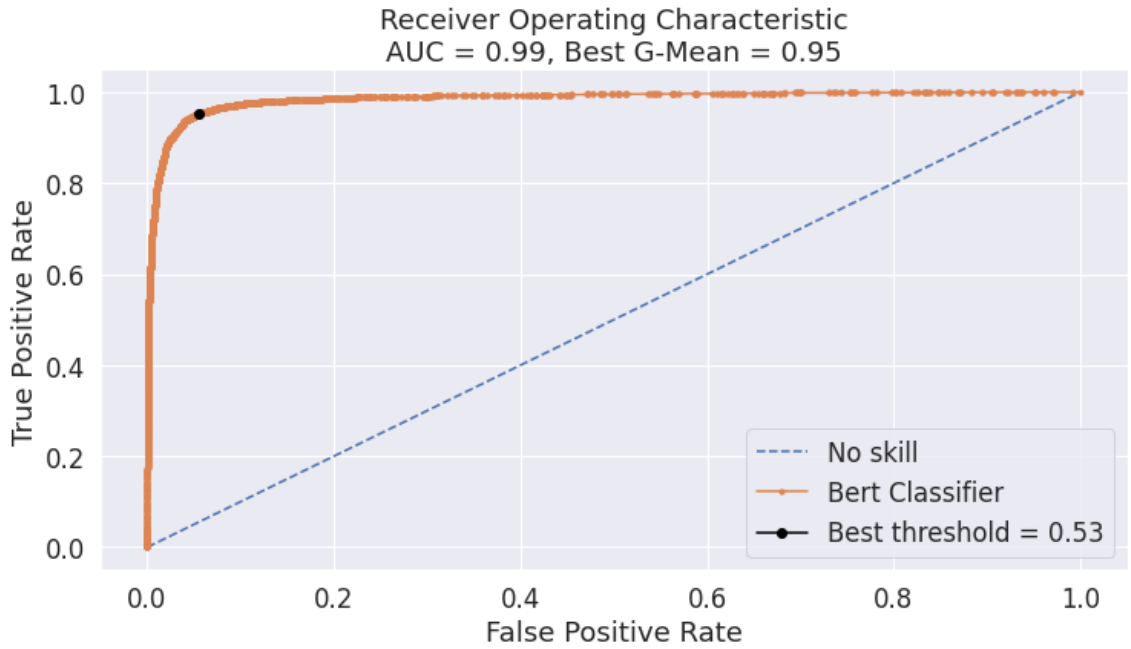


Fig. 6.13. ROC curve of BERT classifier.

An excellent model has AUC (Area Under the Curve) near to the 1 which means it has a good measure of separability. A poor model has AUC near to the 0 which means it has the worst measure of separability. In fact, it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means the model has no class separation capacity whatsoever [134].

To confirm the result we can check it with the Youden's J statistic which is defined as [135]:

$$\begin{aligned} J &= \text{Sensitivity} + \text{Specificity} - 1 = \\ TPR &+ (1 - FPR) - 1 = TPR - FPR \end{aligned} \tag{6.7}$$

```
[125]  1 # get the best threshold
       2 J = tpr - fpr
       3 ix = np.argmax(J)
       4 best_thres = thresholds[ix]
       5 print('Best threshold = %.2f' % (best_thres))

       Best threshold = 0.53
```

Fig. 6.14. Threshold given by Youden's statistic.

Another way to evaluate the skill of a prediction model is with the **Precision-Recall** curve [131] [136].

Precision is a ratio of the number of true positives divided by the sum of the true positives and false positives. It describes how good a model is at predicting the positive class.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.8}$$

As it has been mentioned, **recall** is the same as TPR.

F-Measure or **F1 score** is defined as the harmonic mean of precision (P) and recall (R) [137].

$$F1 = \frac{2PR}{P + R} \tag{6.9}$$

As in the ROC curve, the approach to finding the optimal threshold would be to calculate the F-measure for each threshold and select the largest one. In our case was **0.48** with a score of **0.95**. The precision-recall curve is plotted in figure 6.15.

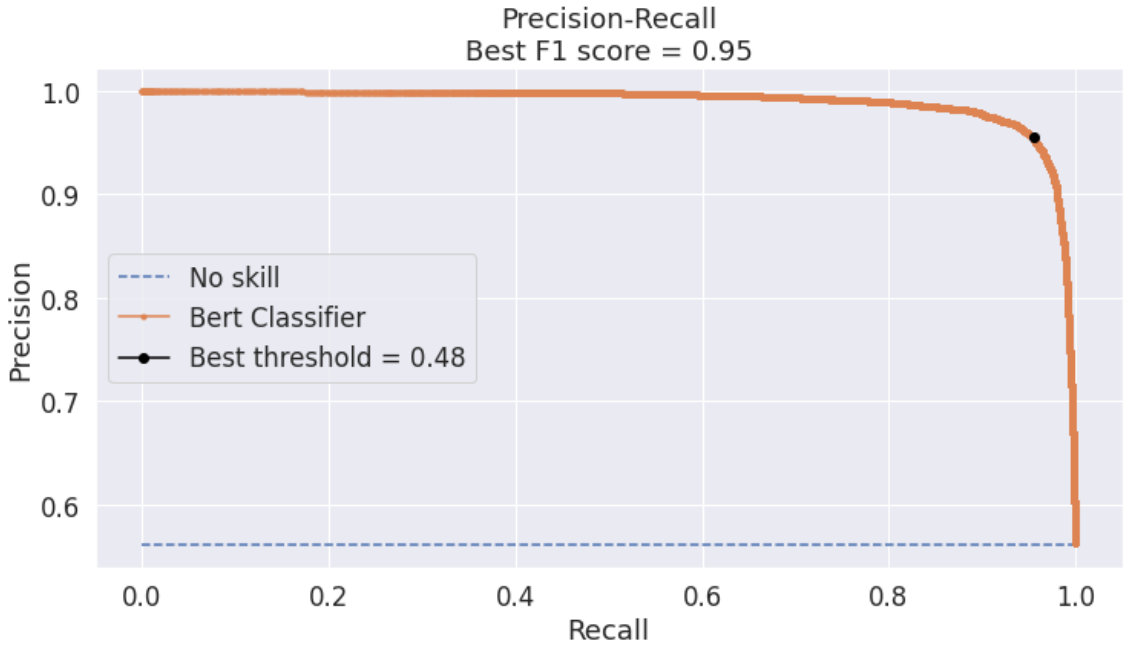


Fig. 6.15. Precision vs. Recall curve of BERT classifier.

A model with perfect skill is depicted as a point at (1,1). A skilful model is represented

by a curve that bows towards (1,1) above the flat line of no skill. Mention thresholds do not have to match, since in each curve the optimization goal is different.

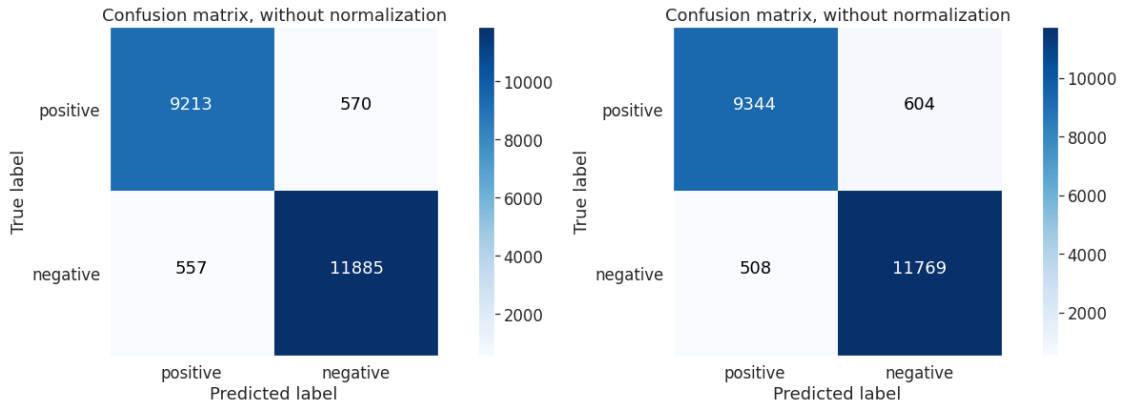
ROC curves should be used when there are roughly equal numbers of observations for each class. Precision-Recall curves should be used when there is a moderate to large class imbalance. The reason for this recommendation is that ROC curves present an optimistic picture of the model on datasets with a class imbalance [138].

Finally, we can predict test samples using a threshold of **0.53**. In figure 6.16 it is presented some samples. It is also checked how results change in the confusion matrix and compare with the previous ones in figure 6.17. In one hand the number of true positives increases. In the other hand the number of true negatives decreases. In spite of that, with 15 hits more, the accuracy hardly improves, taking into account the high number of test samples.

Threshold: 0.53

	sentence	label	real	predicted	positive probability	negative probability
5632	triumph over a scrooge or two	1	positive	positive	0.975746	0.024254
8466	too much of the movie feels contrived , as if the filmmakers were worried the story would n't work without all those gimmicks .	0	negative	negative	0.001697	0.998303
10972	's as comprehensible as any dummies guide , something even non-techies can enjoy .	1	positive	positive	0.997993	0.002007
11799	frighteningly fascinating contradiction	1	positive	positive	0.999133	0.000867
5360	is so bad ,	0	negative	negative	0.001696	0.998304

Fig. 6.16. Some predictions of BERT classifier on STT-2 and 0.53 as threshold.



(a) Confusion matrix with 0.5 threshold. (b) Confusion matrix with 0.53 threshold.

Fig. 6.17. Comparation of confusion matrix changing the threshold.

6.6.2. Final Application for Sentiment Analysis

Finally, in order to get a fast intuiton about the application, it is defined a pipeline to process and classify custom sentences. Some real examples are shown in table 6.9.

Sentence	Sentiment	Probability
After all this process, we can claim this model is super.	Positive	0.98319
	Negative	0.01681
I have learned a lot during my bachelor thesis. Nevertheless, it required a lot of work during these months!	Positive	0.47758
	Negative	0.52242
Next application was completely frustrating. It is difficult to implement.	Positive	0.00177
	Negative	0.99823
After presenting my thesis, I am going to run with my friends to eat a big ice cream. Chocolate is my favourite!	Positive	0.99396
	Negative	0.00604
Honestly, with all these incredible models, I think NLP is going to be a popular trend next years.	Positive	0.99952
	Negative	0.00048

Table 6.9. EXAMPLES OF THE FINAL APPLICATION FOR SENTIMENT ANALYSIS WITH A BERT CLASSIFIER.

7. QUESTION ANSWERING

A more challenged task is question answering, which corresponds to extract the answer to a question from a given context. In this case the model answers the question by taking a substring of the context, not by generating new text.

It is used Stanford Question Answering Dataset (SQuAD) [115], a collection of 100K crowdsourced question/answer pairs. Given a question and a passage from Wikipedia containing the answer, the task is to predict the answer text span in the passage. This corresponds with SQuAD 1.1. In SQuAD 2.0 the problem definition is extended allowing for the possibility that no short answer exists in the provided paragraph, making the problem more realistic.

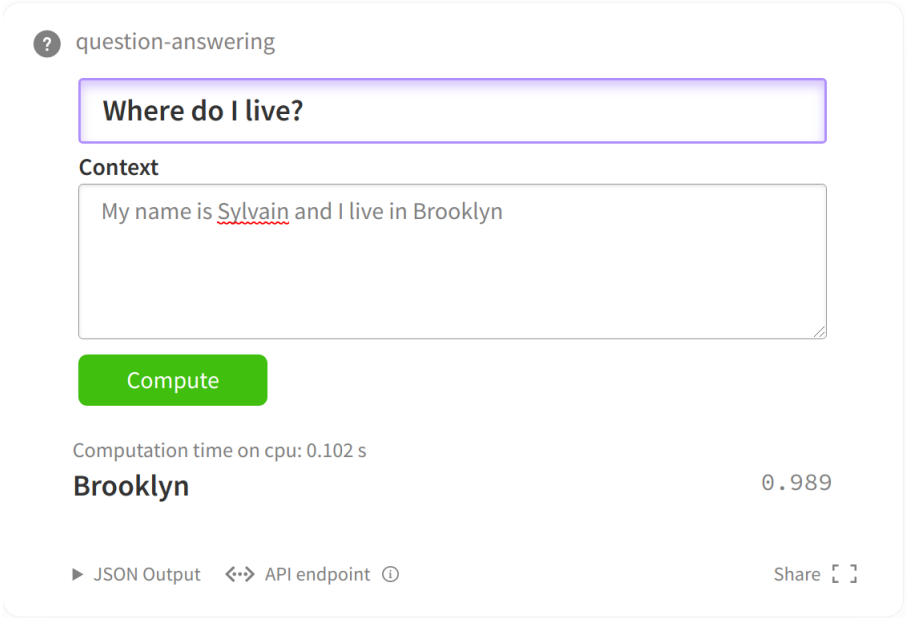


Fig. 7.1. Example of question answering in one of the Hugging Face's models [139].

For this task, tutorial [139] has been used as guide and support to the development of the application.

7.1. Setup

As before, it is employed the GPUs provided by Google Colab. Due to the long training time and the limited resources of Colab in the free version, it was necessary an upgrade. In this case it was assigned a GPU of the pro version: **Tesla V100-SXM2-16GB**.

7.2. Load SQuAD Dataset

It is used Hugging Face Datasets [18] and their API's. Specifying the name of the dataset it is loaded the required data. The original dataset is an object of DatasetDict which is a dictionary containing train and validation samples, with the correspondig **context**, **question** and **answer**. As in the previous task, SQuAD is also used in competitions. In this case, there are not original test data. As before, one third of training set is going to be used as test. In figure 7.2 it is plotted some samples of training set.

Some sentences of train split, which has a length of 58691

	answers	context	id	question	title
0	{'answer_start': [41], 'text': ['Dell 2.0']}	Dell announced a change campaign called "Dell 2.0," reducing the number of employees and diversifying the company's products. While chairman of the board after relinquishing his CEO position, Michael Dell still had significant input in the company during Rollins' years as CEO. With the return of Michael Dell as CEO, the company saw immediate changes in operations, the exodus of many senior vice-presidents and new personnel brought in from outside the company. Michael Dell announced a number of initiatives and plans (part of the "Dell 2.0" initiative) to improve the company's financial performance. These include elimination of 2006 bonuses for employees with some discretionary awards, reduction in the number of managers reporting directly to Michael Dell from 20 to 12, and reduction of "bureaucracy". Jim Schneider retired as CFO and was replaced by Donald Carty, as the company came under an SEC probe for its accounting practices.	570fd59c5ab6b8190039105d	What was the name of Dell's change campaign?	Dell
1	{'answer_start': [381], 'text': ['2004']}	The Times is the originator of the widely used Times Roman typeface, originally developed by Stanley Morison of The Times in collaboration with the Monotype Corporation for its legibility in low-tech printing. In November 2006 The Times began printing headlines in a new font, Times Modern. The Times was printed in broadsheet format for 219 years, but switched to compact size in 2004 in an attempt to appeal more to younger readers and commuters using public transport. The Sunday Times remains a broadsheet.	5705e06c52bb891400689646	In what year did The Times change its broadsheet format to a compact size?	The_Times
2	{'answer_start': [610], 'text': ['by subtracting 1/3 exposure stop']}	The Weston Cadet (model 852 introduced in 1949), Direct Reading (model 853 introduced 1954) and Master III (models 737 and S141.3 introduced in 1956) were the first in their line of exposure meters to switch and utilize the meanwhile established ASA scale instead. Other models used the original Weston scale up until ca. 1955. The company continued to publish Weston film ratings after 1955, but while their recommended values often differed slightly from the ASA film speeds found on film boxes, these newer Weston values were based on the ASA system and had to be converted for use with older Weston meters by subtracting 1/3 exposure stop as per Weston's recommendation. Vice versa, "old" Weston film speed ratings could be converted into "new" Westons and the ASA scale by adding the same amount, that is, a film rating of 100 Weston (up to 1955) corresponded with 125 ASA (as per ASA PH2.5-1954 and before). This conversion was not necessary on Weston meters manufactured and Weston film ratings published since 1956 due to their inherent use of the ASA system; however the changes of the ASA PH2.5-1960 revision may be taken into account when comparing with newer ASA or ISO values.	57264e6f5951b619008f6f71	How were Weston values changed to ASA values?	Film_speed

Fig. 7.2. Some samples of training SQuAD set.

As before, there is the option to remove some data. In table 7.1 it is specified the number of samples per set used in the model's implementation.

Training	Validation	Test
58691	10570	28908

Table 7.1. NUMBER OF SQUAD SAMPLES PER SET USED.

7.3. Preprocessing SQuAD Data

In this task the structure of the input differs from sentiment analysis. Now there are two sentences:

[CLS] Sentence A [SEP] Sentence B

It is employed BertTokenizerFast. This one is a Fast Tokenizer, which according to Hugging Face implies:

1. A significant speed-up in particular when doing batched tokenization.

2. Additional methods to map between the original string (character and words) and the token space.

As before, bert-base-uncased is used as pre-trained model. In spite of being a fast tokenizer, it has the same attributes than the normal option, presented in table 6.4.

One problem is how to manage long documents. The usual truncation process could carry in losing the answer we are looking for. To deal with it, it is truncated the context into features, text shorter than maximum length allowed, and allowed some overlap between them in case the answer lies at the point it is splitted a long context. It is chosen **384** as maximum length of a feature and **128** as the authorized overlap between two parts of the context.

Only the context should be truncated. The tokenizer will return a list of features capped by a certain maximum length, with the overlap. Mention it is necessary to find in which of those features the answer actually is, and where exactly in that feature, the start and end positions.

Once the original datasets are tokenized, it is obtained another DatasetDict, but with different columns than the original one: *attention mask*, used to indicate words from additional padding, *input ids*, tokens as integers, and *token type ids*, indicating 0 or 1, if the token comes from the first sentence (the question) or the second (the context), respectively. Finally start and end positions of the answer.

Some sentences of train split, which has a length of 59315

	attention_mask	end_positions	input_ids	start_positions	token_type_ids
0	[1, ...]	92	[101, 2096, 2002, 2001, 1999, 3905, 2082, 1010, 2054, 4066, 1997, 16841, 2106, 28924, 2490, 1029, 102, 28924, 4114, 13616, 1998, 5500, 14921, 21289, 1996, 12078, 1012, 1999, 2255, 3777, 1010, 2002, 2419, 1037, 10467, 21248, 7795, 1005, 1055, 22965, 2013, 1996, 2142, 5424, 3072, 1012, 2076, 2023, 2027, 3631, 3645, 1997, 1037, 2334, 3309, 5496, 1997, 3529, 6544, 1012, 9105, 1996, 4614, 1005, 3086, 1010, 2027, 10016, 2010, 2155, 2013, 11200, 1012, 28924, 2333, 2000, 28616, 14660, 1010, 2045, 7052, 28616, 14660, 3905, 2082, 1012, 8498, 2010, 3037, 1999, 5424, 8986, 16841, 1010, 2002, 4188, 2000, 3693, 2151, 1997, ...]	90	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...]
1	[1, ...]	154	[101, 2129, 2106, 28924, 2903, 1996, 3956, 1011, 8976, 4736, 2323, 2022, 10395, 1029, 102, 1037, 8050, 2112, 1997, 28924, 1005, 1055, 13165, 2001, 3424, 1011, 19999, 2964, 1012, 2002, 3373, 2008, 1996, 2110, 1997, 3956, 2323, 2025, 4839, 1010, 1998, 2008, 2151, 5424, 12014, 2007, 1996, 5611, 2231, 2001, 1037, 14583, 1997, 1996, 5424, 2111, 1012, 1999, 2312, 2112, 2349, 2000, 2037, 2490, 1997, 3956, 1010, 28924, 26626, 1996, 2142, 2163, 1010, 6195, 1996, 2406, 2000, 2022, 4461, 2923, 1998, 12559, 14083, 2075, 2009, 2004, 1000, 1996, 7861, 5092, 21341, 1997, 4763, 1012, 1000, 8320, 2075, 2114, 5181, 1999, ...]	148	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...]
2	[1, ...]	0	[101, 1999, 2054, 2095, 2106, 1037, 2413, 2932, 6235, 1996, 2224, 1997, 16295, 1029, 102, 15486, 1999, 2605, 2127, 1996, 4329, 1997, 9500, 1012, 2059, 1010, 1999, 1996, 20400, 1010, 2045, 2001, 1037, 12058, 1997, 3037, 1010, 1998, 16295, 2150, 4235, 2109, 1000, 2005, 14271, 2015, 1010, 4257, 15753, 1010, 1998, 1996, 14834, 1997, 20199, 16451, 2015, 1010, 1998, 1999, 2563, 1010, 2070, 2224, 1997, 2009, 2018, 2042, 2081, 1997, 2009, 2005, 2714, 5682, 1000, 1012, 2049, 4125, 1999, 2885, 2001, 1000, 1037, 5573, 9575, 1000, 1010, 2044, 3019, 10042, 2020, 2179, 1000, 1999, 2605, 2012, 9808, 8193, 2078, 1006, ...]	0	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...]

Fig. 7.3. Tokenized examples in SQuAD dataset.

7.4. Generation of the Model

To fine tune on question answering, it is introduced a start vector $S \in \mathbb{R}^H$ and an end vector $E \in \mathbb{R}^H$.

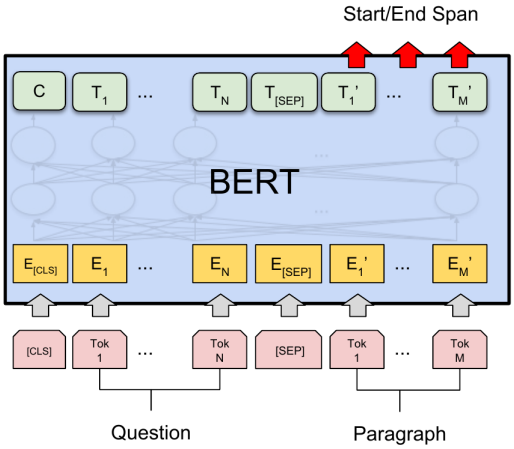


Fig. 7.4. Fine-tune BERT on question answering [9].

To compute the probability of a word i being the start or the end of the answer, it is computed the dot product between S and T_i followed by a softmax over all the words in the sequence.

$$P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}} \tag{7.1}$$

The score for a candidate is defined as $S \cdot T_i + E \cdot T_j$, where i is the start position and j the final position. The maximum scoring span: $\hat{s_{i,j}} = max_{j \geq i} S \cdot T_i + E \cdot T_j$ is used as prediction, where the training objective is the sum of the log-likelihoods of the correct start and end positions.

To extend to SQuAD 2.0, questions which do not have an answer have an span with start and end at the [CLS] token. It is compared the score of the null answer span: $s_{null} = S \cdot C + E \cdot C$ with $\hat{s_{i,j}}$. It is predicted a non-null answer if $\hat{s_{i,j}} > s_{null} + \tau$, where threshold τ is selected on the validation set to maximize F1.

In this task it is employed the model designed by Hugging Face BertForQuestionAnswering.

7.5. Fine-Tune on Question Answering

It is used an instance of Trainer class in order to avoid many lines of code. The most important is the TrainingArguments, which is a class that contains all the attributes to customize the training. It requires one folder name, which will be used to save the checkpoints of the model, and all other arguments are optional. Selected ones for the model are presented in table 7.2.

The evaluation is set to be done at the end of each epoch, an option of Hugging Face API's. Also, it is possible to customize the batch size per GPU core for training and evaluation.

Then it is only needed to pass the arguments and dataset to Trainer. It is also used a data

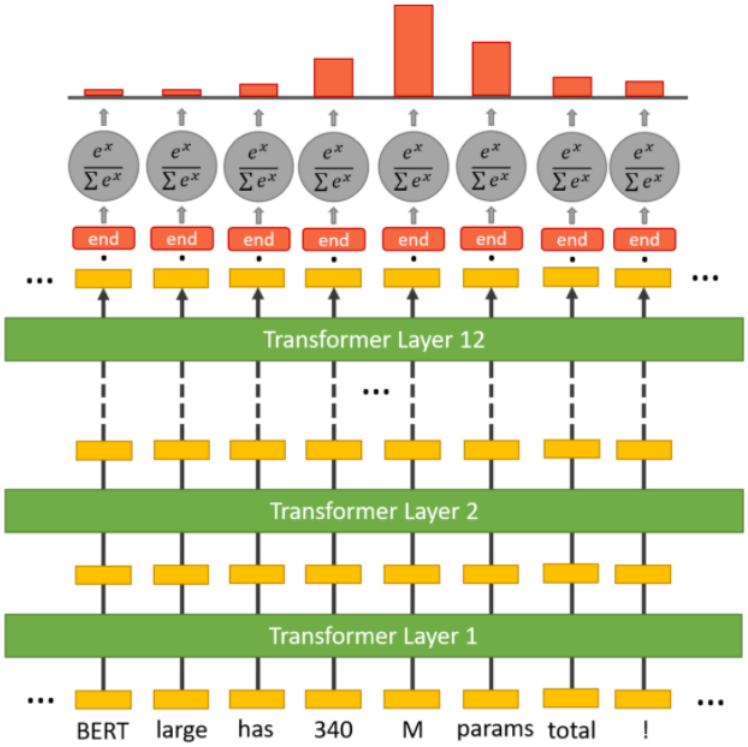


Fig. 7.5. Fine-tune procedure on question answering [140].

Parameter	Value
batch size	16
learning rate	5e-5
epochs	3
epsilon	1e-8
weight decay	0.01

Table 7.2. SELECTED PARAMETERS FOR BERT IN QUESTION ANSWERING.

collator to batch the processed examples together. The tokenizer is used again in order to perfom padding according to the model's preferences: rigth or left and with which token. In order to fine-tune the model it is only required to call the train method for Trainer.

After training, Hugging Face provides a summary of both training and validation losses. It is specified that these ones were calculated per epoch. The saved checkpoints are loaded in order to extract the required information. Mention Hugging Face does not provide the training time per epoch, only total one. Due to that, it is assumed an equal distribution of time in each epoch.

The results are presented in table 7.3 and plotted in figure 7.6. As in the previous task, training loss is decreasing as the validation increases, an overfitting problem. Total training time was **1 hour 23 minutes and 21 seconds**.

Epoch	Training Loss	Validation Loss	Training Time	Validation Time
1	1.3294	1.086206	0:27:47	0:01:16
2	0.7180	1.077027	0:27:47	0:01:16
3	0.4083	1.287688	0:27:47	0:01:16

Table 7.3. BERT FOR QUESTION ANSWERING TRAINING RESULTS.

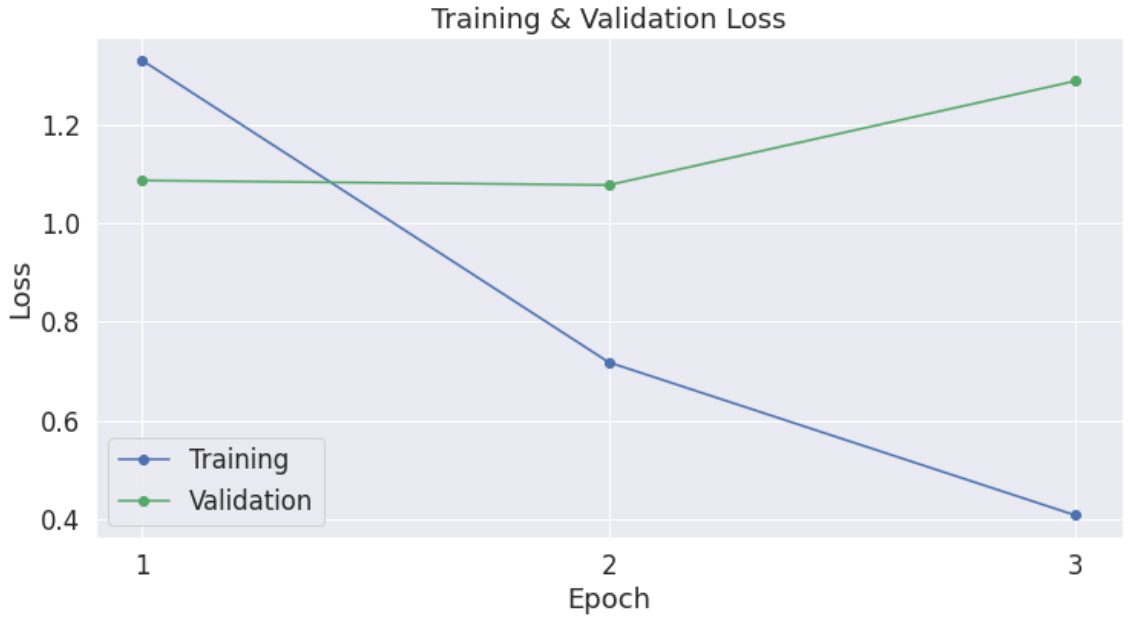


Fig. 7.6. Training vs. validation loss in SQuAD.

7.6. Evaluation of the Model

Once the model is trained, in order to perform the predictions and get an intuition about the result, it is needed to map the predictions back to parts of the context. The model itself predicts logits for the start and end position of the answers. The output of the model is a dict-like object that contains the loss (since it is provided labels), the start and end logits.

There is one logit per feature and token. One idea to predict an answer is to take the index for the maximum of the start logits as a star position and the index of the maximum of the ends logits as an end position. The problem is if this prediction gives us an impossible result, as a greater start position than the end one.

To classify the answer, it is added the start and end logits to obtain a score. Possible answers are limited with a hyper-parameter. Best indices in the start and end logits are picked and gather all the answers this predicts. After checking if each one is valid, they are sorted by their score and it is kept the best one.

Also it is required to check if a given span is inside the context and how to get back the text inside. It is added to the validation features the ID of the example that generates the

feature and the offset mapping to map token indices to character positions.

Finally it is obtained the predictions for all features by using the predict method from Trainer. Results are presentend in table 7.4.

Test Loss	Test Time
1.022919	0:03:49

Table 7.4. SQUAD TEST RESULTS.

It is set *None* in the offset mappings when it corresponds to a part of the question, so it is easy to check if an answer is fully inside the context. It is also eliminated very long answers from the considerations with a hyper-parameter.

It is needed a map between examples and their corresponding features. Then all the answers in all the features generated by a given example are gathered together and picked the best one.

The last bit to deal with is the impossible answer when using SQuAD 2.0. It is required to grab the score for the impossible answer (which has start and end indices corresponding to the index of the CLS token). When one example gives several features, the impossilbe answer is predicted if all the features give a high score to the impossible answer. That is due to one feature could predict the impossible answer just because the answer is not in the part of the context it has access too. Impossible answer is selected when that score is greater than the score of the best non-impossible answer.

Once the features are post processed it is necessary to indicate how to compute metrics from the predictions. It is used a metric which is in charge of compute the corresponding metrics for each model. Metrics computation returns:

- Exact match: the normalized answer exactly match the gold answer. It measures the percentage of predictions that match the truth answer exactly.
- F1: the F-score of predicted tokens versus the gold answer.

Metrics results are in table 7.5. To get a fast intuition on the result, we can plot some predicted sentences in figure 7.7.

Exact match	F1 score
0.67	0.81

Table 7.5. RESULT METRICS FOR BERT ON QUESTION ANSWERING.

	context	question	answers	predicted answers
0	The Human Development Index (HDI) is a composite statistic of life expectancy, education, and income per capita indicators, which are used to rank countries into four tiers of human development. A country scores higher HDI when the life expectancy at birth is longer, the education period is longer, and the income per capita is higher. The HDI was developed by the Pakistani economist Mahbub ul Haq, often framed in terms of whether people are able to "be" and "do" desirable things in their life, and was published by the United Nations Development Programme.	Who developed the HDI?	Mahbub ul Haq	Mahbub ul Haq
1	Great Britain lost Minorca in the Mediterranean to the French in 1756 but captured the French colonies in Senegal in 1758. The British Royal Navy took the French sugar colonies of Guadeloupe in 1759 and Martinique in 1762 as well as the Spanish cities of Havana in Cuba, and Manila in the Philippines, both prominent Spanish colonial cities. However, expansion into the hinterlands of both cities met with stiff resistance. In the Philippines, the British were confined to Manila until their agreed upon withdrawal at the war's end.	What island did Great Britain lose in 1756?	Great Britain lost Minorca in the Mediterranean to the French in 1756	Minorca
2	Although almost all ancient sources relating to crucifixion are literary, the 1968 archeological discovery just northeast of Jerusalem of the body of a crucified man dated to the 1st century provided good confirmatory evidence that crucifixions occurred during the Roman period roughly according to the manner in which the crucifixion of Jesus is described in the gospels. The crucified man was identified as Yehohanan ben Hagkol and probably died about 70 AD, around the time of the Jewish revolt against Rome. The analyses at the Hadassah Medical School estimated that he died in his late 20s. Another relevant archaeological find, which also dates to the 1st century AD, is an unidentified heel bone with a spike discovered in a Jerusalem gravesite, now held by the Israel Antiquities Authority and displayed in the Israel Museum.	What evidence was found that Crucifixion did happen?	the 1968 archeological discovery	gospels

Fig. 7.7. Some predictions of BERT on SQuAD.

7.6.1. Final Application for Question Answering

In order to get a fast intuiton about the application, it is defined a pipeline to process and try to answer to a question from our own text. Some real examples are shown in table 7.6.

Context	Question	Predicted answer
BERT is the best model for NLP, so much better than RNNs... I love it!	Which model do I love?	BERT
In 2016, I started studying telecommunications, it was hard at the beginning, but one year after, 2017, I was sure this was a good option!	When did I start telecommunications?	2016
	When was I sure about studying telecommunications?	2017
Me and my friend are 23 years old. We usually do sport together after studying in the library. Next week we have an important exam, so I am not sure if we could go to the gym.	How old am I?	23
	What do we do after studying?	sport
	What do we have next weeks?	an important exam

Table 7.6. EXAMPLES OF THE FINAL APPLICATION FOR QUESTION ANSWERING WITH BERT.

8. FUTURE OUTLOOK

BERT supossed a critical improvement in NLP field. This model defined a new standard based on pre-training of language models. Considering also the transformer architecture, this approach has been the last trend of latest researchs [141]. Without going into much detail, it is commented some advanced models which succeed BERT.

BERT neglects dependency between the masked positions. This limitation is overcome in **XLNet** [142]. It is an extension of the pre-trained Transformer-XL [143] using an autoregressive method to learn bidirectional contexts by maximizing the expected likelihood over all permutations of the input sequence factorization order. XLNet outperforms BERT on 20 tasks.

Training is computationally expensive and the hyperparameter choices have significant impact on the final results. Facebook AI and the University of Washington researchers analyzed the training of BERT identifying several changes to the training procedure that enhance its performance. These ones are presented in a new model called **RoBERTa** [144] which modifies key hyperparameters, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates. It achieves state-of-the-art results on GLUE [114], RACE [145] and SQuAD [115].

There is a trend in the NLP research of the continuously growing size of the pretrained language models in order to get state-of-the-art results just by using more data and computing power [146]. Some of the latest examples are **Turing-NLG** [147] and **GPT-3** [148].

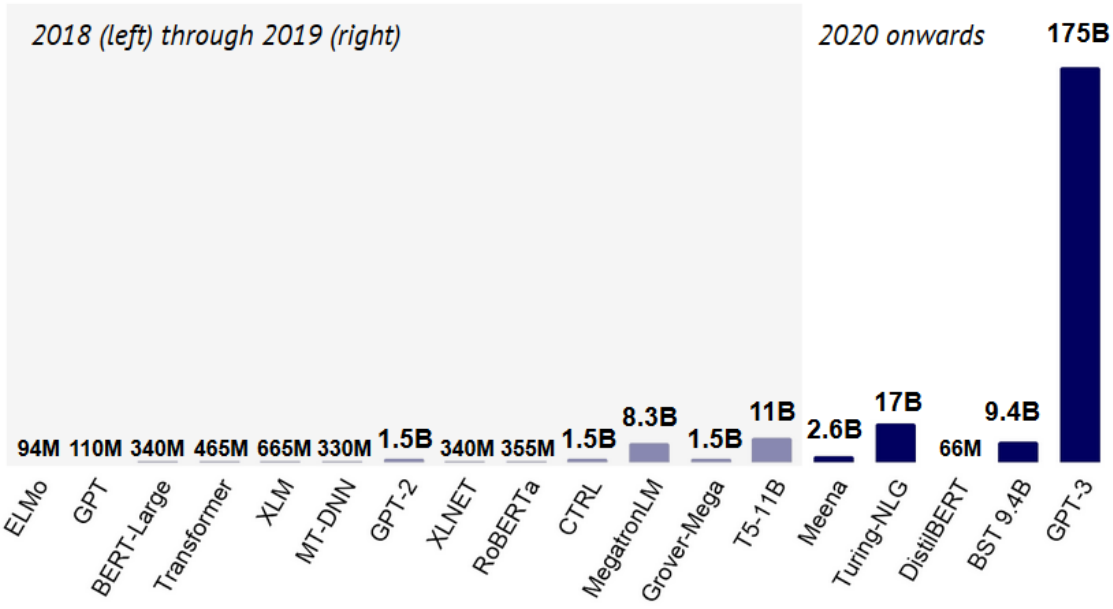


Fig. 8.1. Number of parameters used in latest NLP models [149].

However, this strategy also triggers in memory limitations, longer training time, and sometimes unexpectedly degraded performance. **ALBERT** [150] introduces two techniques to reduce parameters: splitting the embedding matrix into two smaller matrices and using repeating layers split among groups. The model establishes new state-of-the-art results on the GLUE [114], RACE [145], and SQuAD [115] benchmarks while having fewer parameters compared to $\text{BERT}_{\text{LARGE}}$.

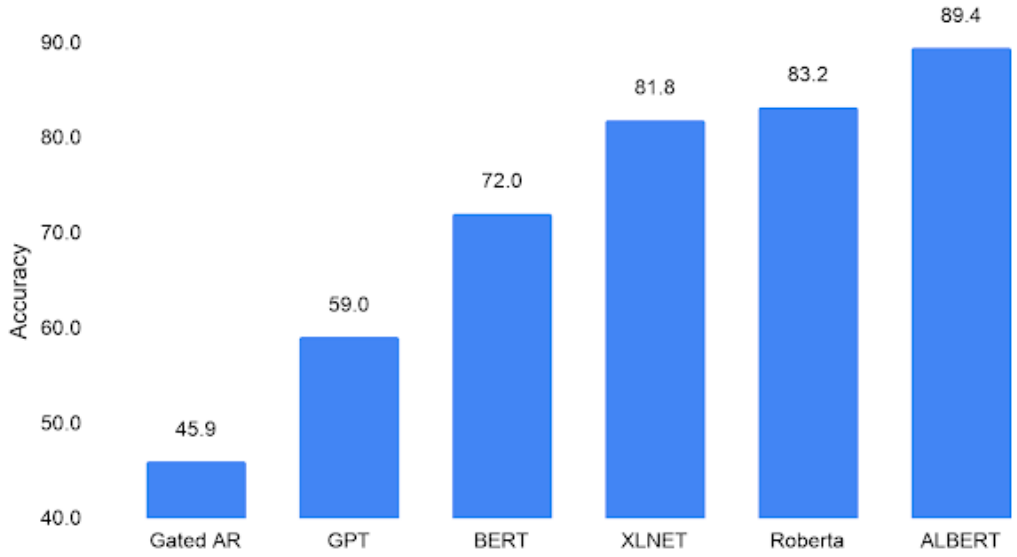


Fig. 8.2. Accuracy results of different models respect ALBERT [151].

Another example is **DistilBERT** [152], a small, fast, cheap and light transformer with 40% less parameters than bert-base-uncased. It runs 60% faster while preserving over 95% of BERT's performances as measured on GLUE. It employs knowledge distillation [153], [154], which is based on training a compact model to reproduce the behaviour of a larger model. The number of layers is reduced by a factor of two, which is a critical factor for computation efficiency [152].

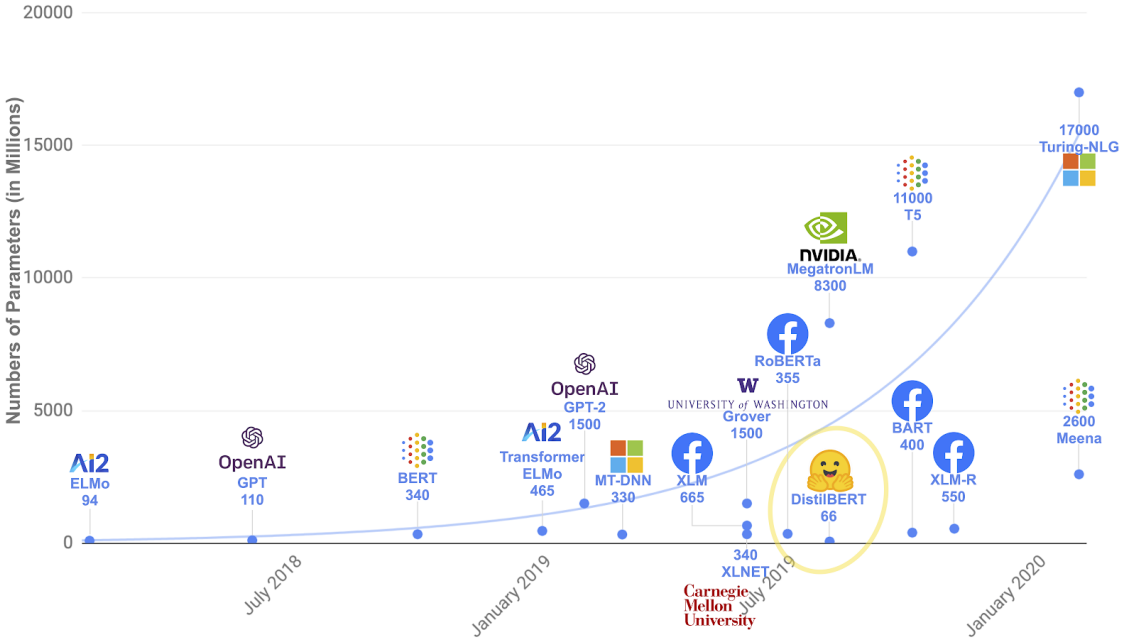


Fig. 8.3. Alternative strategy of DistilBERT against the trend [146] of incremental number of parameters [155].

9. CONCLUSION

9.1. Results

The transformer involved a transition from recurrence and sequential computation to pure attention architectures, providing state-of-the-art models as BERT. Google's algorithm boosts pre-trained language models, since as it has been seen, there a full list of benefits, besides the great improvement in NLP area.

Making use of Hugging Face libraries, it is possible to load the model and preprocess the data according to it. The great feature of BERT is how easily it can be fine-tuned, adding and output layer and training it on a specific task. As it has been shown, the process does not require a huge amount of time.

In 27 minutes of training with 45K samples, it has been implemented a sentiment classifier reaching 95% of accuracy. Mention this result can be improved with several techniques. The clearest ones are trying to avoid overfitting or with hyperparameter searching to find the optimized training arguments. In addition, use a higer number of samples always improves the results.

The same context is applied in question asnwering. Even being both tasks completely different, the models' architecture only differ in the final layer. With a higher training time due to the complexity of the task, in 1 hour and a half it has been developed a model able to answer a question, without the need of belonging to any specified context. Again, hyperparamater searching would improve the results as well as using more training samples.

However, pre-trained language models is a trend in continuos evolution. It has been shown how fast new models overcame BERT's statistics. In spite of that, this model has settled the bases of pre-training for more powerful developments, as well as being an alternative to avoid huge training times in order to get a reliable model.

9.2. Future Work

As it has been mentioned, implemented models can be improved. Main steps to follow are trying to avoid overfitting and hyperparameter searching. In addition to, it can be implemented *Next Sentence Prediction* as a task, in order to replicate the 4 experiments over BERT in the original paper [9].

Multiple models have been published after BERT. It would be interesting testing some of these and compare their performance, as well as studying the differences in the architecture and code implementation. For example, DistilBERT [152] from Hugging Face, is an

interesting option to replicate question answering.

This idea could be a follow-up for this project, showing how fast NLP models can improve with a few number of changes due to the pre-trained base. This work is an option for the topic of my Master's thesis next year.

10. LEGISLATIVE AND REGULATORY FRAMEWORK

Due to the use of massive data, it is critical to know the regulatory framework which involves the topic. The legislation about data storage is very different around the world. For this reason, it is considered the most common use cases, taking into account the European and spanish legislation.

10.1. General Data Protection Regulation

Regarding the European Union, General Data Protection Regulation (GDPR) [156] on the protection of natural persons with regard to the processing of personal data and on the free movement of such data. It is applied across EU members since May 25^{th}, 2018 and replaces the outdated Data Protection Directive from 1998.

In article 5, it is collected the main principles regarding the processing of personal data. In one hand the interested party must consent the use of his or her data and always be informed about it. On the other hand, data treatment must be carried out according to current regulation.

It is also mentioned, in order to collect data there must be a justified purpose and only use the data which is strictly necessary. In addition, the conservation period must be also restricted to the use time.

Finally, it ensures the data protection, in order to avoid its loss, destruction or unauthorized use.

10.2. Organic Law on Protection of Personal Data and Guarantee of Digital Rights

Regarding Spain, the latest law for data protection and privacy is the Organic Law 3/2018 of December 5 on Protection of Personal Data (LOPD) and Guarantee of Digital Rights (GDD) [157].

This law is adapted to previous mentioned General Data Protection Regulation (GDPR), so the fundamental right to the data protection is regulated according to both. The scope of application involves to any processing of personal data. It includes total or parcial use, as well as automated or non automated.

Last, mention the **Spanish Data Protection Agency** which oversees the compliance with the legal provisions on the protection of personal data.

10.3. Intellectual Property

During the project implementation it has been used Python 3.8 and PyTorch, a deep learning framework. Both are open-source and free-to-use, so there is not any regulatory problem in the development. Mention also Google Colab, which do not have any legislative restriction in our scope as well as the free version of PyCharm.

The used datasets, GLUE [114] and SQuAD [115], are published and accessible by everyone, as well as Hugging Face models [17] and datasets [18].

As it has been mentioned in section 1.4, the notebook, the memory and other material are published in my github (github.com/ion-bueno/bert-from-inside).

11. SOCIAL AND ECONOMIC ENVIRONMENT

11.1. Social Impact

Artificial Intelligence (AI) is becoming more prevalent in our lives, gaining relevance in exponential way. AI impact has increased significantly in the last year [158]. Due to the critical advances in NLP, there are some points to take into account.

11.1.1. Ethical Challenges

Ethical challenges of AI applications are a controversial topic nowadays. In [158] it is exposed that address these challengues and build responsible systems is critical in order to reach fair innovations. However, they remark how the field lacks benchmarks to measure the relationship between broader societal discussions about technology development and the development of the technology itself.

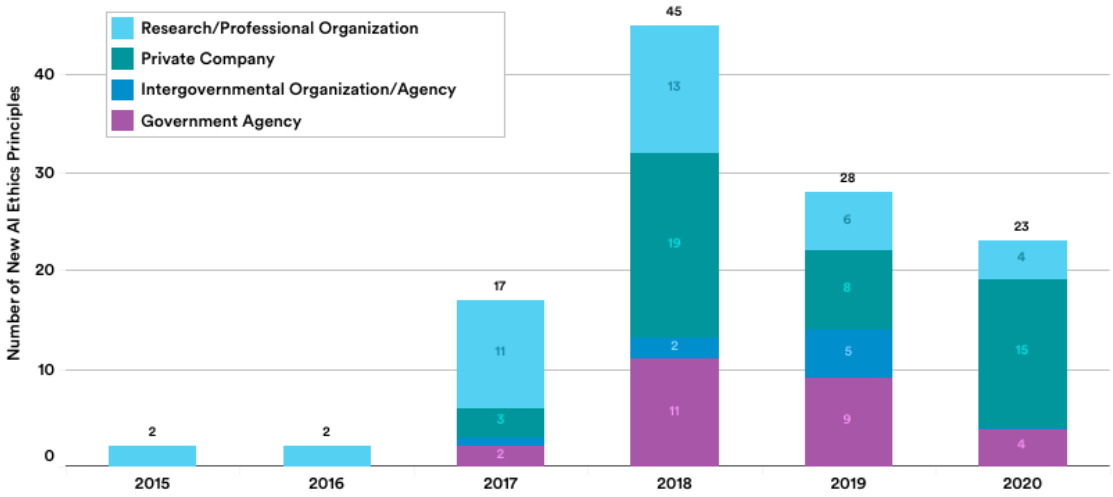


Fig. 11.1. AI ethics principles by organization type [158].

Focusing on the topic of the project, in [159] it is identified a number of social implications of NLP and discussed their ethical significance. Language is an instrument of power, with influence in multiple sectors [160], [161]. The social impact factors of NLP are derived from the mutual relationships between language, society and individual.

Some points to focus are demographic bias in training samples, which leads to the **exclusion** of determined regions. In contrast, **overgeneralization** is the side-effect. The problem resides in relying on models producing false positives, which may lead to bias confirmation.

Another important issue is the creation of biases due to **topic overexposure** [162]. De-

termined characteristics could be associated to a certain type of people or the difficulty to process a language to an abnormal group. Nevertheless, **underexposure** can negatively impact evaluation of small languages.

Even if there is not intention of any harm and previous issues are addressed, there can be unintended consequences [163]. Some of them could be censorship thanks to text classification, generation of fake news or degrading of non-standard languages due to educational applications.

11.1.2. Research

NLP has grown into a major research discipline. The number of publications has increased dramatically in the past 20 years with major powers racing to invest in the topic. Google manages to dominate this space while Microsoft holds a respectable second position and CMU is the the top publishing university.

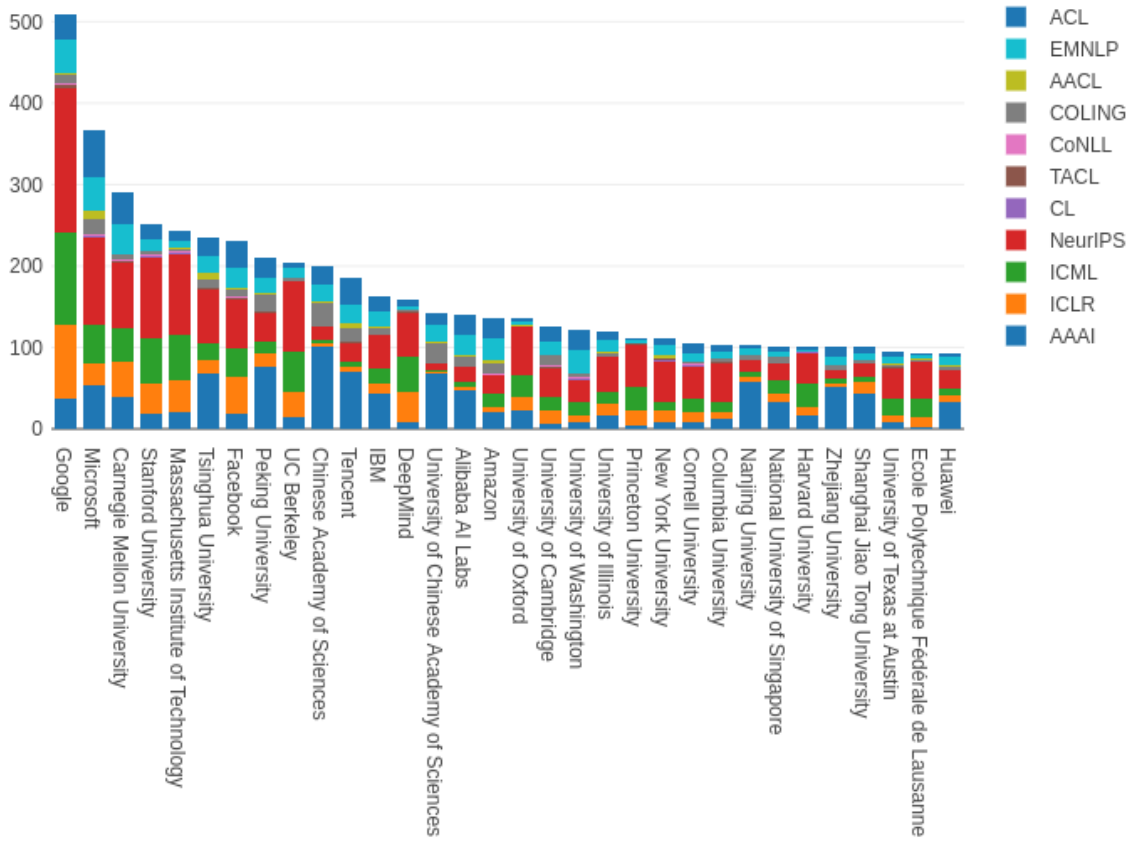


Fig. 11.2. Published NLP papers during 2020 [164].

However the highest proportion of papers comes from academic institutions. The second source depends on the country, being popular corporate-affiliated research or the government [165].

11.1.3. Education

In this part there is not a distinction to NLP, since this area is usually mixed with other AI contents in educational level.

World's top universities have increased their investment in AI education over the past four years. In the last four academic years the number of courses that teach students the skills necessary to build or deploy a practical AI model, on the undergraduate and graduate levels, has increased by 102.9% and 41.7%, respectively [158].

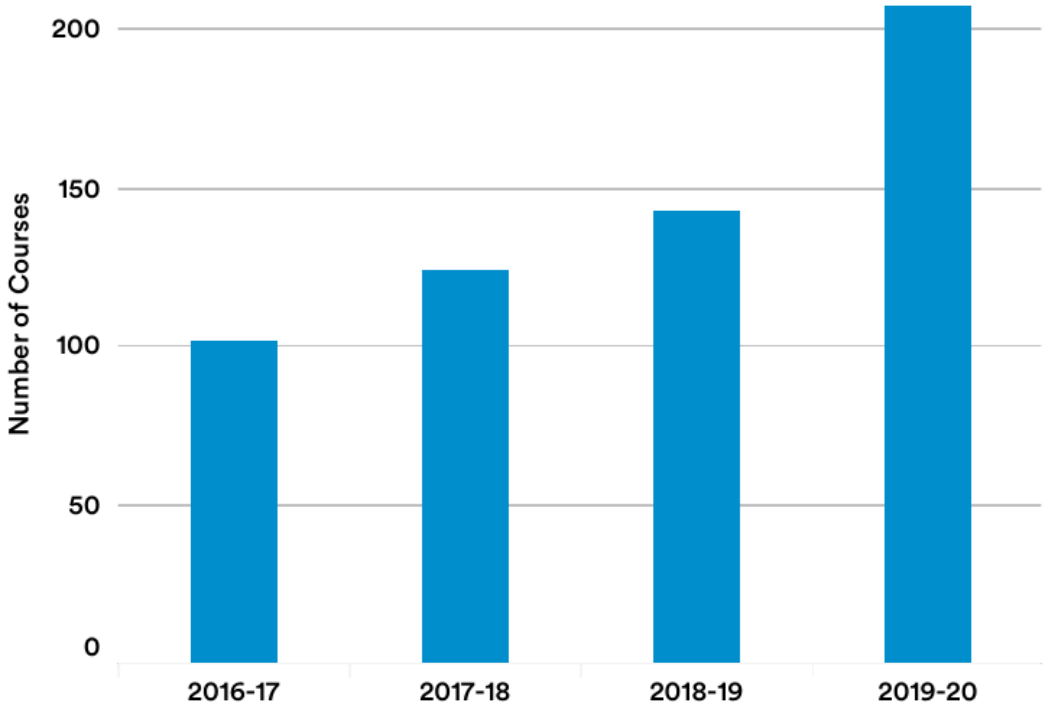


Fig. 11.3. Number of undergraduate courses that teach AI skills [158].

Following the research trend, the United States, China and Europe have also the leadership in the educational area. With great difference, the United States is the country with more graduate students, being many of them from countries outside Europe, as it is shown in figure 11.4.

11.2. Economic Impact

The total investment in AI increased by 40% in 2020 relative to 2019 (figure 11.5). The pandemic of COVID 19 has resulted into industry consolidation and increased Mergers and Acquisitions (M&A) activity, driving up the total corporate investment against small businesses [158].

NLP help organizations turn unstructured data into insights and make information easy for customers. For example, IBM published a report [166] analyzing some key findings

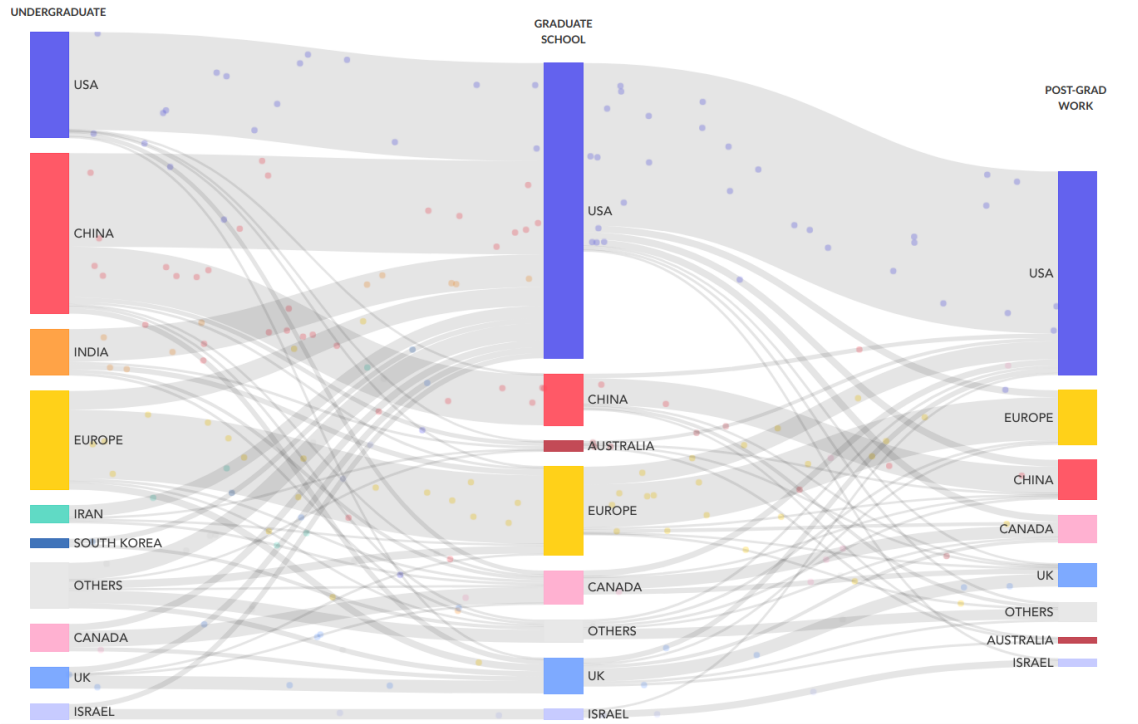


Fig. 11.4. AI careers demographic paths [165].

of their NLP tools. They reflected a return on investment (ROI) of 383% and a net present value NPV of 4.86M dolars.

In the state of AI report of 2020 [149] more applications with a great economic impact were highlighted. Compliance officers are overloaded with manual research using keywords, while NLP techniques can cover up to 85% of the risk data in all key geographies. Machine translation is also used to generate multilingual training data for financial crime classification, reducing lead time from 20 weeks (for English) to less than two weeks.

Going into more detail with the thesis's topic, mention BERT incorporation to Google and Microsoft's Bing search query. In only 12 months from open source publications, the model has been implemented in large scale production. It is worth mentioning the use in production of open source Hugging Face models [17] and datasets [18] from more than 1,000 companies. At July 2020 there were 5M pip installs and more than 2500 community transformer models trained in over 164 languages by 430 contributors.

11.3. Project Planning

First months were intended to literature research and reading. From the basis of RNNs and advanced derivatives to self-attention and transformer, followed by testing repositories and external code for the transformer and BERT.

From March to June, it starts the implementation in a notebook, as well as collecting information for the memory. Mention some text of the notebook was included in the

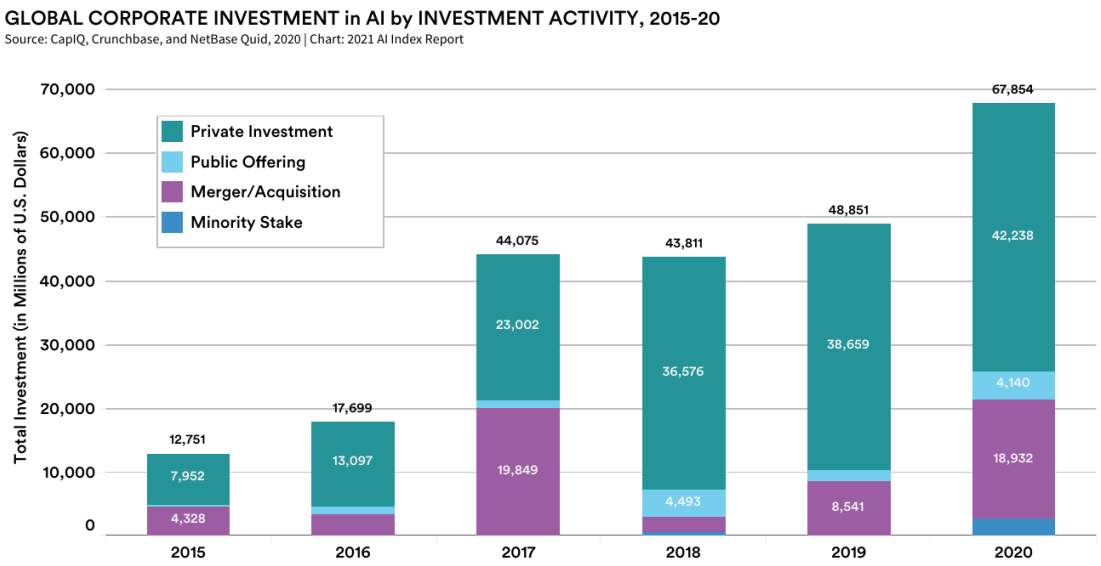


Fig. 11.5. Global corporate investment in AI [158].

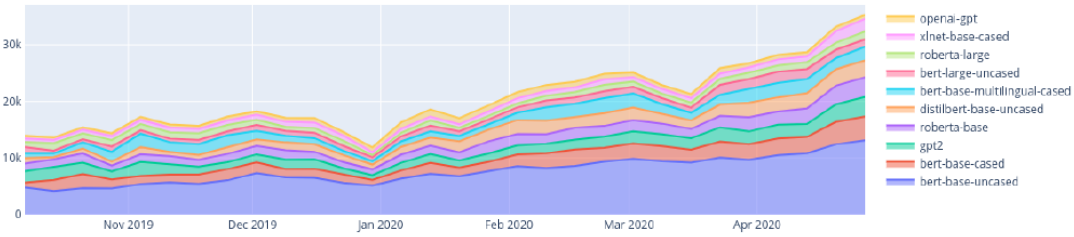


Fig. 11.6. Average daily unique downloads of the most downloaded Hugging Face pretrained models, Oct. 2019 to May 2020 [17].

memory, so that is why both were developed almost in parallel time.

The last interval time corresponds with the review of the notebook and memory. In figure 11.7 it is plotted the progress of the project during 9 months, from September 2020 to June 2021.

11.4. Project Budet

This thesis can be thought as a first approach for a later research project. The team would be composed by the main researcher and a graduate engineer. The estimated time corresponds with 9 months, with following costs:

- Personal computer which satisfies the computational requirements. One proposal would be a Dell Inspiron 13 7000.
- Computation needs. Necessary hardware as GPUs or a subscription to the pro version of Google Colab during the months of development.

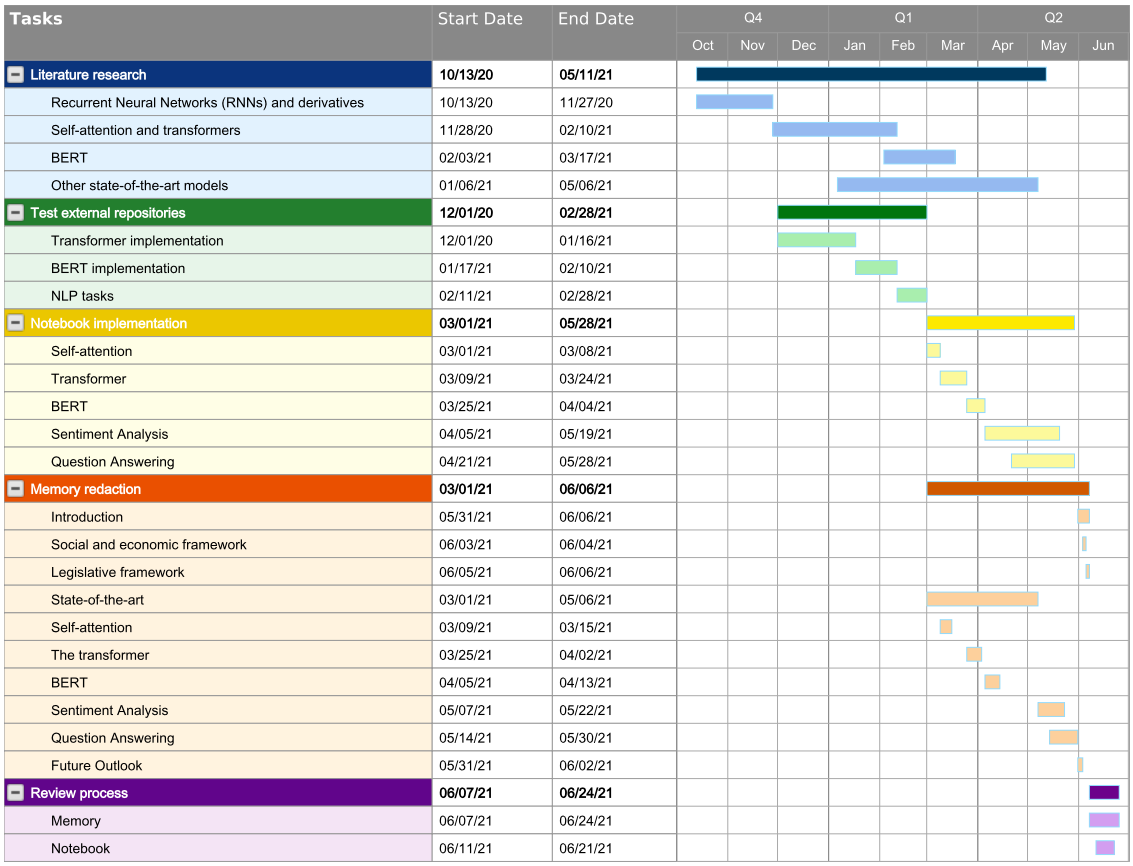


Fig. 11.7. Gantt chart of the project.

- Salaries, for the graduate engineer at full time and the main researcher. This last one will be covered by the university, while the graduate engineer depends on the university and country. In this case it is used the UC3M remunerations.
- Other costs as transportation and memory publishing.

The proposal project budgget is presented in table 11.1.

Items	Unit	Cost per unit (€)	Total cost (€)
Personal computer	1	1000	1000
Google Colab Pro	3 months	10	30
UC3M graduate engineer	9 months	1,602.97€	14,426.73
Other costs	1	1000	1000
Total			**16,456.73**

Table 11.1. PROJECT BUDGET OF THE RESEARCH PROJECT.

BIBLIOGRAPHY

[1] N. Zheng *et al.*, "Predicting covid-19 in china using hybrid AI model," *IEEE Transactions on Cybernetics*, vol. 50, no. 7, pp. 2891–2904, Jul. 2020. [Online]. Available: https://doi.org/10.1109/TCYB.2020.2990162 (visited on Jun. 18, 2021).

[2] P. Wang, X. Zheng, G. Ai, D. Liu, and B. Zhu, "Time series prediction for the epidemic trends of covid-19 using the improved lstm deep learning method: Case studies in Russia, Peru and Iran," *Chaos, Solitons & Fractals*, vol. 140, p. 110 214, Nov. 2020. [Online]. Available: https://doi.org/10.1016/j.chaos.2020.110214 (visited on Jun. 18, 2021).

[3] P. Bose, S. Roy, and P. Ghosh, "A comparative NLP-based study on the current trends and future directions in covid-19 research," *IEEE Access*, vol. 9, pp. 78 341–78 355, May 2021. [Online]. Available: https://doi.org/10.1109/ACCESS.2021.3082108 (visited on Jun. 18, 2021).

[4] V. Kieuvongngam, B. Tan, and Y. Niu, "Automatic text summarization of covid-19 medical research articles using BERT and GPT-2," *Computing Research Repository*, vol. abs/2006.01997, arXiv:2006.01997, Jun. 2020. [Online]. Available: https://arxiv.org/abs/2006.01997 (visited on Jun. 18, 2021).

[5] H. Jelodar, Y. Wang, R. Orji, and S. Huang, "Deep sentiment classification and topic discovery on novel coronavirus or covid-19 online discussions: NLP using LSTM recurrent neural network approach," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 10, pp. 2733–2742, Jun. 2020. [Online]. Available: https://doi.org/10.1109/JBHI.2020.3001216 (visited on Jun. 18, 2021).

[6] I. Li *et al.*, "What are we depressed about when we talk about covid-19: Mental health analysis on tweets using natural language processing," in *Artificial Intelligence XXXVII*, 1st ed. Cambridge, United Kingdom: Springer International Publishing, 2020, pp. 358–370.

[7] A. Clouder. (Mar. 12, 2020). Fighting coronavirus with technology: Another breakthrough for alibaba in nlp research, Alibaba Cloud, [Online]. Available: https://www.alibabacloud.com/blog/fighting-coronavirus-with-technology-another-breakthrough-for-alibaba-in-nlp-research_595973 (visited on Jun. 18, 2021).

[8] W. Wang *et al.*, "StructBERT: Incorporating language structures into pre-training for deep language understanding," *Computing Research Repository*, vol. abs/1908.04577, arXiv:1908.04577, Sep. 2019. [Online]. Available: https://arxiv.org/abs/1908.04577 (visited on Jun. 18, 2021).

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *Computing Research Repository*, vol. abs/1810.04805, arXiv:1810.04805, May 2019. [Online]. Available: https://arxiv.org/abs/1810.04805 (visited on Jun. 18, 2021).

[10] P. Nayak. (Oct. 25, 2019). Understanding searches better than ever before, Google, [Online]. Available: https://blog.google/products/search/search-language-understanding-bert/ (visited on Jun. 18, 2021).

[11] M. de Alwis. (Aug. 10, 2020). Google's BERT update: A new way with words, ADAPT, [Online]. Available: https://www.adaptworldwide.com/insights/2020/googles-bert-update (visited on Jun. 18, 2021).

[12] B. Csutoras. (Nov. 26, 2019). BERT explained: What you need to know about Google's new algorithm, Search Engine Journal, [Online]. Available: https://www.searchenginejournal.com/bert-explained-what-you-need-to-know-about-googles-new-algorithm/337247/#close (visited on Jun. 18, 2021).

[13] H. Wang, D. Can, A. Kazemzadeh, F. Bar, and S. Narayanan, "A system for real-time twitter sentiment analysis of 2012 us presidential election cycle," in *Proceedings of the ACL 2012 system demonstrations*, Jeju Island, Korea, Jul. 1, 2012. [Online]. Available: https://www.aclweb.org/anthology/P12-3020.

[14] A. Ortigosa, J. M. Martín, and R. M. Carro, "Sentiment analysis in Facebook and its application to e-learning," *Computers in Human Behavior*, vol. 31, pp. 527–541, Aug. 2013. [Online]. Available: https://doi.org/10.1016/j.chb.2013.05.024 (visited on Jun. 18, 2021).

[15] Y. Liu, J. Bian, and E. Agichtein, "Predicting information seeker satisfaction in community question answering," in *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, United States, Jul. 20, 2008. [Online]. Available: https://doi.org/10.1145/1390334.1390417.

[16] A. Ben Abacha and P. Zweigenbaum, "MEANS: A medical question-answering system combining NLP techniques and semantic web technologies," *Information Processing & Management*, vol. 51, no. 5, pp. 570–594, Sep. 15. [Online]. Available: http://dx.doi.org/10.1016/j.ipm.2015.04.006 (visited on Jun. 18, 2021).

[17] T. Wolf *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *Computing Research Repository*, vol. abs/1910.03771, arXiv:1910.03771, Jul. 2020. [Online]. Available: https://arxiv.org/abs/1910.03771 (visited on Jun. 18, 2021).

[18] Q. Lhoest *et al.* (Jun. 8, 2021). Huggingface/datasets: 1.8.0, Hugging Face.

[19] G. G. Chowdhury, "Natural language processing," *Annual Review of Information Science and Technology*, vol. 37, no. 1, pp. 51–89, Jan. 2005. [Online]. Available: https://doi.org/10.1002/aris.1440370103 (visited on Jun. 18, 2021).

[20] G. Salton, J. Allan, and C. Buckley, "Approaches to passage retrieval in full text information systems," in *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, United States, Jul. 1, 1993. [Online]. Available: https://doi.org/10.1145/160688.160693.

[21] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: An introduction," *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544–551, Sep. 2011. [Online]. Available: https://doi.org/10.1136/amiajnl-2011-000464 (visited on Jun. 18, 2021).

[22] L. Dirac. (Dec. 3, 2019). LSTM is dead. long live transformers! [Online]. Available: https://www.youtube.com/watch?v=S27pHKBEp30&t=256s (visited on 2021).

[23] L. Wu, S. C. H. Hoi, and N. Yu, "Semantics-preserving bag-of-words models and applications," *IEEE Transactions on Image Processing*, vol. 19, no. 7, pp. 1908–1920, Oct. 2009. [Online]. Available: https://doi.org/10.1145/1631058.1631064 (visited on Jun. 18, 2021).

[24] X. Wang, A. McCallum, and X. Wei, "Topical N-grams: Phrase and topic discovery, with an application to information retrieval," in *Seventh IEEE International Conference on Data Mining*, Omaha, United States, Oct. 28, 2007. [Online]. Available: https://doi.org/10.1109/ICDM.2007.86.

[25] G. Sidorov, F. Velasquez, E. Stamatatos, A. Gelbukh, and L. Chanona-Hernández, "Syntactic N-grams as machine learning features for natural language processing," *Expert Systems with Applications*, vol. 41, no. 3, pp. 853–860, Aug. 2013. [Online]. Available: https://doi.org/10.1016/j.eswa.2013.08.015 (visited on Jun. 18, 2021).

[26] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, 1st ed. Kitchener, Canada: D2L, 2020. [Online]. Available: https://d2l.ai.

[27] A. Vaswani *et al.*, "Attention is all you need," *Computing Research Repository*, vol. abs/1706.03762, arXiv:1706.03762, Dec. 2017. [Online]. Available: https://arxiv.org/abs/1706.03762 (visited on Jun. 18, 2021).

[28] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *The Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, Mar. 2003. [Online]. Available: https://dl.acm.org/doi/10.5555/944919.944966 (visited on Jun. 18, 2021).

[29] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, Cambridge, United States, Dec. 8, 2014. [Online]. Available: https://dl.acm.org/doi/10.5555/2969033.2969173.

[30] R. Socher, E. Huang, J. Pennington, A. Ng, and C. Manning, "Dynamic pooling and unfolding recursive autoencoders for paraphrase detection," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, Red Hook, United States, Dec. 12, 2011. [Online]. Available: https://dl.acm.org/doi/10.5555/2986459.2986549.

[31] A. Jagannatha and H. Yu, "Structured prediction models for RNN based sequence labeling in clinical text," *Computing Research Repository*, vol. abs/1608.00612, arXiv:1608.00612, Aug. 2016. [Online]. Available: https://arxiv.org/abs/1608.00612 (visited on Jun. 18, 2021).

[32] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, Doha, Qatar, Oct. 1, 2014. [Online]. Available: http://dx.doi.org/10.3115/v1/D14-1179.

[33] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," May 7, 2015. [Online]. Available: http://arxiv.org/abs/1409.0473.

[34] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-RNN: Deep learning on spatio-temporal graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, United States, Dec. 12, 2016. [Online]. Available: https://doi.org/10.1109/CVPR.2016.573.

[35] F. Wang and D. Tax, "Survey on the attention based RNN model and its applications in computer vision," *Computing Research Repository*, vol. abs/1601.06823, arXiv:1601.06823, Jan. 2016. [Online]. Available: https://arxiv.org/abs/1601.06823 (visited on Jun. 18, 2021).

[36] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and Intelligent Laboratory Systems*, vol. 39, no. 1, pp. 43–62, Feb. 1997. [Online]. Available: https://doi.org/10.1016/S0169-7439(97)00061-0 (visited on Jun. 18, 2021).

[37] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *Computing Research Repository*, vol. abs/1412.3555, arXiv:1412.3555, Dec. 2014. [Online]. Available: https://arxiv.org/abs/1412.3555 (visited on Jun. 18, 2021).

[38] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, *et al.*, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Networks*, 1st ed. Ann Arbor, United States: Wiley-IEEE Press, 2001, pp. 237–243.

[39] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent," in *Backpropagation: Theory, architectures, and applications*, 1st ed. Hillsdale, United States: Lawrence Erlbaum Associates Inc, 1995, pp. 433–486.

[40] G. Bird and M. E. Polivoda, "Backpropagation through time for networks with long-term dependencies," *Computing Research Repository*, vol. abs/2103.15589, arXiv:2103.15589, Apr. 2021. [Online]. Available: https://arxiv.org/abs/2103.15589 (visited on Jun. 18, 2021).

[41] S. Y. Fei-Fei Li Justin Johnson, *Lecture 10: Recurrent neural networks*, 2018. [Online]. Available: http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture10.pdf.

[42] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994. [Online]. Available: https://doi.org/10.1109/72.279181 (visited on Jun. 18, 2021).

[43] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, Canada, Oct. 21, 2013. [Online]. Available: https://doi.org/10.1109/ICASSP.2013.6639349.

[44] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, Madison, United States, Jun. 28, 2011. [Online]. Available: https://dl.acm.org/doi/10.5555/3104482.3104612.

[45] J. Martens, "Deep learning via hessian-free optimization," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Madison, United States, Jun. 21, 2010. [Online]. Available: https://dl.acm.org/doi/10.5555/3104322.3104416.

[46] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735 (visited on Jun. 18, 2021).

[47] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *Computing Research Repository*, vol. abs/1409.1259, arXiv:1409.1259, Oct. 2014. [Online]. Available: https://arxiv.org/abs/1409.1259 (visited on Jun. 18, 2021).

[48] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE international conference on acoustics, speech and signal processing*, Vancouver, Canada, May 26, 2013. [Online]. Available: https://doi.org/10.1109/ICASSP.2013.6638947.

[49] T. Stérin, N. Farrugia, and V. Gripon, "An intrinsic difference between vanilla rnns and gru models," in *The Ninth International Conference on Advanced Cognitive Technologies and Applications*, Athens, Greece, Feb. 19, 2017. [Online]. Available: https://dna.hamilton.ie/tsterin/static/pdf/An_Intrinsic_Difference_Between_Vanilla_GRU_Sterin_Farrugia_Gripon.pdf.

[50] CodeEmporium. (Jan. 13, 2020). Transformer neural networks - explained! (attention is all you need), [Online]. Available: https://www.youtube.com/watch?v=TQQlZhbC5ps&list=PLTl9hO2Oobd_bzXUpzKMKA3liq2kj6LfE (visited on Jun. 18, 2021).

[51] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, 1st ed. Hershey, United States: IGI global, 2010, pp. 242–264.

[52] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," in *2nd International Conference on Learning Representations*, Banff, Canada, Apr. 14, 2014. [Online]. Available: http://arxiv.org/abs/1312.6026.

[53] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, Canada, May 26, 2013. [Online]. Available: https://doi.org/10.1109/ICASSP.2013.6638947.

[54] L. Mou, P. Ghamisi, and X. X. Zhu, "Deep recurrent neural networks for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 7, pp. 3639–3655, Apr. 2017. [Online]. Available: https://doi.org/10.1109/TGRS.2016.2636241 (visited on Jun. 18, 2021).

[55] S. Venugopalan *et al.*, "Translating videos to natural language using deep recurrent neural networks," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Denver, United States, May 1, 2015. [Online]. Available: http://dx.doi.org/10.3115/v1/N15-1173.

[56] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997. [Online]. Available: https://doi.org/10.1109/78.650093 (visited on Jun. 18, 2021).

[57] P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri, "Exploiting the past and the future in protein secondary structure prediction," *Bioinformatics*, vol. 15, no. 11, pp. 937–946, Nov. 1999. [Online]. Available: https://doi.org/10.1093/bioinformatics/15.11.937 (visited on Jun. 18, 2021).

[58] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, Aug. 2005. [Online]. Available: https://doi.org/10.1016/j.neunet.2005.06.042 (visited on Jun. 18, 2021).

[59] P. Baldi, S. Brunak, P. Frasconi, G. Pollastri, and G. Soda, "Bidirectional dynamics for protein secondary structure prediction," in *Sequence Learning*, 1st ed. Berlin, Germany: Springer, 2001, pp. 80–104.

[60] J. Chen and N. S. Chaudhari, "Capturing long-term dependencies for protein secondary structure prediction," in *International Symposium on Neural Networks*, Dalian, China, Aug. 19, 2004. [Online]. Available: https://doi.org/10.1007/978-3-540-28648-6_79.

[61] M. Schuster, *On supervised learning from sequential data with applications for speech recognition*, Ikoma, Japan, 1999. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.7060&rep=rep1&type=pdf (visited on Jun. 18, 2021).

[62] T. Fukada, M. Schuster, and Y. Sagisaka, "Phoneme boundary estimation using bidirectional recurrent neural networks and its applications," *Systems and Computers in Japan*, vol. 30, no. 4, pp. 20–30, Mar. 1999. [Online]. Available: https://doi.org/10.1002/(SICI)1520-684X(199904)30:4%3C20::AID-SCJ3%3E3.0.CO;2-E (visited on Jun. 18, 2021).

[63] Y. Wu *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *Computing Research Repository*, arXiv:1609.08144, Sep. 2016. [Online]. Available: https://arxiv.org/abs/1609.08144 (visited on Jun. 18, 2021).

[64] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *Computing Research Repository*, vol. abs/1508.04025, arXiv:1508.04025, Aug. 2015. [Online]. Available: https://arxiv.org/abs/1508.04025 (visited on Jun. 18, 2021).

[65] O. Kuchaiev and B. Ginsburg, "Factorization tricks for LSTM networks," *Computing Research Repository*, vol. abs/1703.10722, arXiv:1703.10722, Feb. 2018. [Online]. Available: https://arxiv.org/abs/1703.10722 (visited on Jun. 18, 2021).

[66] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations*, San Diego, United States, May 7, 2015. [Online]. Available: https://arxiv.org/abs/1409.1556.

[67] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," *Computing Research Repository*, vol. abs/1306.0543, arXiv:1306.0543, Jun. 2013. [Online]. Available: https://arxiv.org/abs/1306.0543 (visited on Jun. 18, 2021).

[68] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386 (visited on Jun. 18, 2021).

[69] S. K. Esser *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the national academy of sciences*, vol. 113, no. 41, pp. 11 441–11 446, Aug. 2016. [Online]. Available: https://doi.org/10.1073/pnas.1604850113 (visited on Jun. 18, 2021).

[70] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, Hawai, United States, Jul. 21, 2017. [Online]. Available: https://doi.org/10.1109/CVPR.2017.195.

[71] D. Eigen, M. Ranzato, and I. Sutskever, "Learning factored representations in a deep mixture of experts," in *2nd International Conference on Learning Representations*, Banff, Canada, Apr. 14, 2014. [Online]. Available: https://arxiv.org/abs/1312.4314.

[72] A. Davis and I. Arel, "Low-rank approximations for conditional feedforward computation in deep neural networks," in *2nd International Conference on Learning Representations*, Banff, Canada, Apr. 14, 2017. [Online]. Available: https://arxiv.org/abs/1312.4461.

[73] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *Computing Research Repository*, vol. abs/1308.3432, arXiv:1308.3432, Aug. 2013. [Online]. Available: https://arxiv.org/abs/1308.3432 (visited on Jun. 18, 2021).

[74] N. Shazeer *et al.*, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *Computing Research Repository*, vol. abs/1701.06538, arXiv:1701.06538, Jun. 2017. [Online]. Available: https://arxiv.org/abs/1701.06538 (visited on Jun. 18, 2021).

[75] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, Mar. 1991. [Online]. Available: https://doi.org/10.1162/neco.1991.3.1.79 (visited on Jun. 18, 2021).

[76] M. Johnson *et al.*, "Google's multilingual neural machine translation system: Enabling zero-shot translation," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 339–351, Aug. 2017. [Online]. Available: https://doi.org/10.1162/tacl_a_00065 (visited on Jun. 18, 2021).

[77] R. Collobert, S. Bengio, and Y. Bengio, "A parallel mixture of SVMs for very large scale problems," *Neural computation*, vol. 14, no. 5, pp. 1105–1114, May 2002. [Online]. Available: %5Ctoday.

[78] B. Yao, D. Walther, D. Beck, and L. Fei-fei, "Hierarchical mixture of classification experts uncovers interactions between brain regions," in *Advances in Neural Information Processing Systems*, Vancouver, Canada, Dec. 7, 2009. [Online]. Available: https://dl.acm.org/doi/10.5555/2984093.2984337.

[79] R. Aljundi, P. Chakravarty, and T. Tuytelaars, "Expert gate: Lifelong learning with a network of experts," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, United States, Jul. 21, 2017. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2017.753.

[80] G. Loye. (Sep. 15, 2019). Attention mechanism, FloydHub Blog, [Online]. Available: https://blog.floydhub.com/attention-mechanism/ (visited on Jun. 18, 2021).

[81] A. M. Treisman and G. Gelade, "A feature-integration theory of attention," *Cognitive Psychology*, vol. 12, no. 1, pp. 97–136, Jan. 1980. [Online]. Available: https://doi.org/10.1016/0010-0285(80)90005-5 (visited on Jun. 18, 2021).

[82] M. J. Er, Y. Zhang, N. Wang, and M. Pratama, "Attention pooling-based convolutional neural network for sentence modelling," *Information Sciences*, vol. 373, pp. 388–403, Dec. 2016. [Online]. Available: https://doi.org/10.1016/j.ins.2016.08.084 (visited on Jun. 18, 2021).

[83] Y. Kim, C. Denton, L. Hoang, and A. M. Rush, "Structured attention networks," *Computing Research Repository*, vol. abs/1702.00887, arXiv:1702.00887, Feb. 2017. [Online]. Available: https://arxiv.org/abs/1702.00887 (visited on Jun. 18, 2021).

[84] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal: Association for Computational Linguistics, Sep. 1, 2015. [Online]. Available: http://dx.doi.org/10.18653/v1/D15-1044.

[85] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," *Computing Research Repository*, vol. abs/1705.04304, arXiv:1705.04304, May 2017. [Online]. Available: https://arxiv.org/abs/1705.04304 (visited on Jun. 18, 2021).

[86] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, United States, Nov. 1, 2016. [Online]. Available: http://dx.doi.org/10.18653/v1/D16-1053.

[87] Z. Lin *et al.*, *A structured self-attentive sentence embedding*, Mar. 2017. [Online]. Available: https://arxiv.org/abs/1703.03130 (visited on Jun. 18, 2021).

[88] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model for natural language inference," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, United States, Nov. 1, 2016. [Online]. Available: http://dx.doi.org/10.18653/v1/D16-1244.

[89] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *Computing Research Repository*, vol. abs/1602.02410, arXiv:1602.02410, Feb. 2016. [Online]. Available: https://arxiv.org/abs/1602.02410 (visited on Jun. 19, 2021).

[90] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," *Computing Research Repository*, vol. abs/1503.08895, arXiv:1503.08895, Nov. 2015. [Online]. Available: https://arxiv.org/abs/1503.08895 (visited on Jun. 19, 2021).

[91] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *3rd International Conference on Learning Representations*, San Diego, United States, May 7, 2015. [Online]. Available: https://arxiv.org/abs/1410.3916.

[92] J. Weston *et al.*, "Towards AI-complete question answering: A set of prerequisite toy tasks," in *4th International Conference on Learning Representations*, San Juan, Puerto Rico, May 2, 2016. [Online]. Available: https://arxiv.org/abs/1502.05698.

[93] Ł. Kaiser and I. Sutskever, "Neural GPUs learn algorithms," in *4th International Conference on Learning Representations*, San Juan, Puerto Rico, May 2, 2016. [Online]. Available: https://arxiv.org/abs/1511.08228.

[94] Ł. Kaiser and S. Bengio, "Can active memory replace attention?" *Computing Research Repository*, vol. abs/1610.08613, arXiv:1610.08613, Mar. 2017. [Online]. Available: https://arxiv.org/abs/1610.08613 (visited on Jun. 19, 2021).

[95] N. Kalchbrenner *et al.*, "Neural machine translation in linear time," *Computing Research Repository*, vol. abs/1610.10099, arXiv:1610.10099, Mar. 2017. [Online]. Available: https://arxiv.org/abs/1610.10099 (visited on Jun. 19, 2021).

[96] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *Proceedings of the 34th International Conference on Machine Learning*, Sydney, Australia, Aug. 6, 2017. [Online]. Available: https://arxiv.org/abs/1705.03122.

[97] L. Svensson. (Nov. 27, 2020). A series of videos on transformers, [Online]. Available: https://www.youtube.com/playlist?list=PLDw5cZwIToCvXLVY2bSqt7F2gu8y-Rqje (visited on Jun. 18, 2021).

[98] P. Kar and H. Karnick, "Random feature maps for dot product kernels," in *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, San Diego, United States, Jun. 8, 2015. [Online]. Available: https://arxiv.org/abs/1201.6530.

[99] R. Karim. (Nov. 18, 2019). Illustrated: Self-attention, Towards data science, [Online]. Available: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a (visited on Jun. 19, 2021).

[100] B. Hoover, H. Strobelt, and S. Gehrmann, "ExBERT: A visual analysis tool to explore learned representations in transformer models," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Online, Jul. 5, 2020. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-demos.22.

[101] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, "OpenNMT: Open-source toolkit for neural machine translation," in *Proceedings of ACL 2017, System Demonstrations*, Vancouver, Canada, Jul. 1, 2017. [Online]. Available: https://www.aclweb.org/anthology/P17-4012.

[102] A. Graves, "Generating sequences with recurrent neural networks," *Computing Research Repository*, vol. abs/1308.0850, arXiv:1308.0850, Jun. 2014. [Online]. Available: https://arxiv.org/abs/1308.0850 (visited on Jun. 19, 2021).

[103] O. Press and L. Wolf, "Using the output embedding to improve language models," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, Valencia, Spain, Apr. 1, 2017. [Online]. Available: https://www.aclweb.org/anthology/E17-2025.

[104] D. Britz, A. Goldie, M.-T. Luong, and Q. Le, "Massive exploration of neural machine translation architectures," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark, Sep. 1, 2017. [Online]. Available: http://dx.doi.org/10.18653/v1/D17-1151.

[105] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Las Vegas, United States, Jun. 27, 2016. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90.

[106] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *Computing Research Repository*, vol. abs/1607.06450, arXiv:1607.06450, Jul. 2016. [Online]. Available: https://arxiv.org/abs/1607.06450 (visited on Jun. 19, 2021).

[107] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 929–1958, Jan. 2014. [Online]. Available: https://dl.acm.org/doi/10.5555/2627435.2670313 (visited on Jun. 19, 2021).

[108] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, Aug. 1, 2016. [Online]. Available: http://dx.doi.org/10.18653/v1/P16-1162.

[109] M. E. Peters *et al.*, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, New Orleans, United States, Jun. 1, 2018. [Online]. Available: http://dx.doi.org/10.18653/v1/N18-1202.

[110] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. (Jun. 11, 2018). Improving language understanding with unsupervised learning, OpenAI, [Online]. Available: https://openai.com/blog/language-unsupervised/ (visited on Jun. 18, 2021).

[111] M. Schuster and K. Nakajima, "Japanese and korean voice search," in *International Conference on Acoustics, Speech and Signal Processing*, Kyoto, Japan, Mar. 25, 2012. [Online]. Available: https://doi.org/10.1109/ICASSP.2012.6289079.

[112] Y. Zhu *et al.*, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, Dec. 7, 2015. [Online]. Available: https://doi.org/10.1109/ICCV.2015.11.

[113] CodeEmporium. (May 4, 2020). BERT neural network - explained! [Online]. Available: https://www.youtube.com/watch?v=xI0HHN5XKDo&list=PLTl9hO2Oobd_bzXUpzKMKA3liq2kj6LfE&index=3 (visited on Jun. 18, 2021).

[114] A. Wang *et al.*, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Brussels, Belgium, Nov. 1, 2018. [Online]. Available: http://dx.doi.org/10.18653/v1/W18-5446.

[115] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, United States, Nov. 1, 2016. [Online]. Available: http://dx.doi.org/10.18653/v1/D16-1264.

[116] R. Zellers, Y. Bisk, R. Schwartz, and Y. Choi, "SWAG: A large-scale adversarial dataset for grounded commonsense inference," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, Oct. 1, 2018. [Online]. Available: http://dx.doi.org/10.18653/v1/D18-1009.

[117] R. Socher *et al.*, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, Seattle, United States, Oct. 1, 2013. [Online]. Available: https://www.aclweb.org/anthology/D13-1170.

[118] S. Gugger. (Oct. 19, 2020). Fine-tuning a model on a text classification task, Hugging Face, [Online]. Available: https://github.com/huggingface/notebooks/blob/master/examples/text_classification.ipynb (visited on Jun. 21, 2021).

[119] C. Tran. (). Tutorial: Fine tuning BERT for sentiment analysis, Skim AI, [Online]. Available: https://skimai.com/fine-tuning-bert-for-sentiment-analysis/ (visited on Jun. 20, 2021).

[120] C. McCormick and N. Ryan. (Jul. 22, 2019). Bert fine-tuning tutorial with pytorch, Chris McCormick AI, [Online]. Available: http://mccormickml.com/2019/07/22/BERT-fine-tuning/ (visited on Jun. 20, 2021).

[121] C. Horan. (Jan. 28, 2020). Tokenizers: How machines read, FloydHub Blog, [Online]. Available: https://blog.floydhub.com/tokenization-nlp/ (visited on Jun. 21, 2021).

[122] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, Aug. 1, 2016. [Online]. Available: http://dx.doi.org/10.18653/v1/P16-1162.

[123] L. Le Cam, "Maximum likelihood: An introduction," *International Statistical Review/Revue Internationale de Statistique*, vol. 58, no. 2, pp. 153–171, Aug. 1990. [Online]. Available: https://doi.org/10.2307/1403464 (visited on Jun. 20, 2021).

[124] A. Rowshan. (Feb. 9, 2021). Datasets & dataloaders, PyTorch, [Online]. Available: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html (visited on Jun. 20, 2021).

[125] J. Alammar. (Dec. 3, 2018). The illustrated BERT, ELMo, and co. (how NLP cracked transfer learning), Jalammar Github, [Online]. Available: http://jalammar.github.io/illustrated-bert/ (visited on Jun. 21, 2021).

[126] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations*, San Diego, United States, May 7, 2015. [Online]. Available: https://arxiv.org/abs/1412.6980.

[127] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen, "Assessing the accuracy of prediction algorithms for classification: An overview," *Bioinformatics*, vol. 16, no. 5, pp. 412–424, May 2000. [Online]. Available: https://doi.org/10.1093/bioinformatics/16.5.412 (visited on Jun. 20, 2021).

[128] J. Brownlee. (Dec. 22, 2020). A gentle introduction to cross-entropy for machine learning, Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/cross-entropy-for-machine-learning/ (visited on Jun. 20, 2021).

[129] ——, (Aug. 12, 2019). Embrace randomness in machine learning, Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/randomness-in-machine-learning/ (visited on Jun. 20, 2021).

[130] ——, (Jan. 5, 2021). A gentle introduction to threshold-moving for imbalanced classification, Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/ (visited on Jun. 20, 2021).

[131] ——, (Aug. 13, 2018). How to use ROC curves and precision-recall curves for classification in python, Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/ (visited on Jun. 20, 2021).

[132] T. Fawcett, "Roc graphs: Notes and practical considerations for researchers," *Machine learning*, vol. 31, pp. 1–38, Jan. 2014. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.4749&rep=rep1&type=pdf (visited on Jun. 20, 2021).

[133] J. Akosa, "Predictive accuracy: A misleading performance measure for highly imbalanced data," 2017. [Online]. Available: http://support.sas.com/resources/papers/proceedings17/0942-2017.pdf (visited on Jun. 20, 2021).

[134] S. Narkhede. (Jun. 26, 2018). Understanding auc - roc curve, Towards data science, [Online]. Available: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (visited on Jun. 20, 2021).

[135] V. Bewick, L. Cheek, and J. Ball, "Statistics review 13: Receiver operating characteristic curves," *Critical care*, vol. 8, no. 6, pp. 1–5, Nov. 2004. [Online]. Available: https://ccforum.biomedcentral.com/articles/10.1186/cc3000 (visited on Jun. 20, 2021).

[136] V. Raghavan, P. Bollmann, and G. S. Jung, "A critical investigation of recall and precision as measures of retrieval system performance," *ACM Transactions on Information Systems*, vol. 7, pp. 205–229, Jul. 1989. [Online]. Available: https://doi.org/10.1145/65943.65945 (visited on Jun. 20, 2021).

[137] D. M. W. Powers, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation," *Computing Research Repository*, vol. abs/2010.16061, arXiv:2010.16061, Oct. 2020. [Online]. Available: https://arxiv.org/abs/2010.16061 (visited on Jun. 20, 2021).

[138] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," in *Proceedings of the 23rd international conference on Machine learning*, New York, United States, Jun. 25, 2006. [Online]. Available: https://doi.org/10.1145/1143844.1143874.

[139] A. Bacciu. (Dec. 15, 2020). Fine-tuning a model on a question-answering task, Hugging Face, [Online]. Available: https://github.com/huggingface/notebooks/blob/master/examples/question_answering.ipynb (visited on Jun. 20, 2021).

[140] C. McCormick. (Mar. 10, 2020). Question answering with a fine-tuned BERT, Chris McCormick AI, [Online]. Available: https://mccormickml.com/2020/03/10/question-answering-with-a-fine-tuned-BERT/ (visited on Jun. 21, 2021).

[141] M. Yao. (May 11, 2021). 10 leading language models for NLP in 2021, TOPBOTS, [Online]. Available: https://www.topbots.com/leading-nlp-language-models-2020/ (visited on Jun. 20, 2021).

[142] Z. Yang *et al.*, "XLNet: Generalized autoregressive pretraining for language understanding," *Computing Research Repository*, vol. abs/1906.08237, arXiv:1906.08237, Jan. 2020. [Online]. Available: https://arxiv.org/abs/1906.08237 (visited on Jun. 20, 2021).

[143] Z. Dai *et al.*, "Transformer-XL: Attentive language models beyond a fixed-length context," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, Jul. 1, 2019. [Online]. Available: http://dx.doi.org/10.18653/v1/P19-1285.

[144] Y. Liu *et al.*, "RoBERTa: A robustly optimized BERT pretraining approach," *Computing Research Repository*, vol. abs/1907.11692, arXiv:1907.11692, Jul. 2019. [Online]. Available: https://arxiv.org/abs/1907.11692 (visited on Jun. 20, 2021).

[145] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy, "RACE: Large-scale reading comprehension dataset from examinations," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark, Sep. 1, 2017. [Online]. Available: http://dx.doi.org/10.18653/v1/D17-1082.

[146] A. Rogers. (Jul. 22, 2019). How the transformers broke NLP leaderboards, Hacking Semantics, [Online]. Available: https://hackingsemantics.xyz/2019/leaderboards/ (visited on Jun. 20, 2021).

[147] C. Rosset. (Feb. 13, 2020). Turing-NLG: A 17-billion-parameter language model by microsoft, Microsoft Research Blog, [Online]. Available: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/ (visited on Jun. 20, 2021).

[148] T. B. Brown *et al.*, "Language models are few-shot learners," *Computing Research Repository*, vol. abs/2005.14165, arXiv:2005.14165, Jul. 2020. [Online]. Available: https://arxiv.org/abs/2005.14165 (visited on Jun. 20, 2021).

[149] N. Benaich and H. Ian, *State of AI report 2020*, 2020. [Online]. Available: https://www.stateof.ai/.

[150] Z. Lan *et al.*, "ALBERT: A lite BERT for self-supervised learning of language representations," *Computing Research Repository*, vol. abs/1909.11942, arXiv:1909.11942, Feb. 2020. [Online]. Available: https://arxiv.org/abs/1909.11942 (visited on Jun. 20, 2021).

[151] S. Radu and L. Zhenzhong. (Dec. 20, 2019). ALBERT: A lite BERT for self-supervised learning of language representations, Google AI Blog, [Online]. Available: https://ai.googleblog.com/2019/12/albert-lite-bert-for-self-supervised.html (visited on Jun. 20, 2021).

[152] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," *Computing Research Repository*, vol. abs/1910.01108, arXiv:1910.01108, Mar. 2020. [Online]. Available: https://arxiv.org/abs/1910.01108 (visited on Jun. 20, 2021).

[153] C. Buciluundefined, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, United States, Aug. 20, 2006. [Online]. Available: https://doi.org/10.1145/1150402.1150464.

[154] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *Computing Research Repository*, vol. abs/1503.02531, arXiv:1503.02531, Mar. 2015. [Online]. Available: https://arxiv.org/abs/1503.02531 (visited on Jun. 20, 2021).

[155] C. Pierric, S. Victor, and A. Moi. (May 18, 2020). How Hugging Face achieved a 2x performance boost for question answering with distilbert in node.js, Hugging Face, [Online]. Available: https://blog.tensorflow.org/2020/05/how-hugging-face-achieved-2x-performance-boost-question-answering.html (visited on Jun. 20, 2021).

[156] European Union, *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*, May 4, 2016. [Online]. Available: http://data.europa.eu/eli/reg/2016/679/oj/eng.

[157] Government of Spain, *Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales*, Dec. 6, 2018. [Online]. Available: https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673&p=20210527&tn=1.

[158] D. Zhang *et al.*, "The AI index 2021 annual report," Standford University, Stanford, United States, Tech. Rep., May 2021.

[159] D. Hovy and S. L. Spruit, "The social impact of natural language processing," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, Aug. 1, 2016. [Online]. Available: http://dx.doi.org/10.18653/v1/P16-2096.

[160] K. Laskowski, "Modeling norms of turn-taking in multi-party conversation," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Stroudsburg, United States, Jul. 11, 2010. [Online]. Available: https://dl.acm.org/doi/10.5555/1858681.1858783.

[161] D. Bracewell and M. Tomlinson, "The language of power and its cultural influence," in *Proceedings of COLING 2012: Posters*, Mumbai, India, Oct. 1, 2012. [Online]. Available: https://www.aclweb.org/anthology/C12-2016.pdf.

[162] A. Tversky and D. Kahneman, "Availability: A heuristic for judging frequency and probability," *Cognitive psychology*, vol. 5, no. 2, pp. 207–232, Jan. 1973. [Online]. Available: https://doi.org/10.1016/0010-0285(73)90033-9 (visited on Jun. 20, 2021).

[163] H. Jonas, *The imperative of responsibility: In search of an ethics for the technological age*, 1st ed. Chicago, United States: University of Chicago press, 1985.

[164] M. Rei. (Feb. 11, 2021). ML and NLP publications in 2020, Marek Rei Blog, [Online]. Available: https://www.marekrei.com/blog/ml-and-nlp-publications-in-2020/ (visited on Jun. 20, 2021).

[165] *The global AI talent tracker*, 2020. [Online]. Available: https://macropolo.org/digital-projects/the-global-ai-talent-tracker/.

[166] M. Sarah and M. Connor, "The total economic impact of IBM watson natural language processing (NLP) solutions," Forrester, Tech. Rep., Feb. 2021.