Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Deep Question Answering: A New Teacher For DistilBERT

Relatore:

Chiar.mo Prof.
Fabio Tamburini

Correlatore:

Chiar.mo Prof.
Philipp Cimiano

Presentata da:
Simone Preite

Sessione III
Anno Accademico 2018/2019

Non rimandare a domani quello che potresti fare oggi!

Sommario

Nel vasto ambito del Natural Language Processing (NLP), letteralmente tradotto come elaborazione del linguaggio naturale, sono stati proposti, nel corso del tempo, diversi modelli e tecnologie utili ad aggiungere questa capacità ai calcolatori.

Lungo questo lavoro andremo ad esplorare quali sono gli strumenti disponibili oggigiorno, nello specifico vedremo i Transformer utilizzati all'interno del modello BERT [9], creato da Google, e di alcuni modelli derivati da esso. Questo modello è al momento lo "stato dell'arte" per diversi problemi di NLP.

Andremo, in questa tesi, a focalizzarci sul Question Answering (QA), ovvero l'abilità di cercare automaticamente risposte a domande poste in linguaggio naturale. Il dataset di riferimento sarà SQuAD v2 ma verrà presentato anche un ulteriore dataset sperimentale di nome OLP, entrambi verranno successivamente descritti nel capitolo 4.

Il principale obiettivo di questo lavoro di tesi era quello di sperimentare i benifici ottenibili intervenendo sul livello di question answering. Ci si è ispirati al lavoro prodotto dal gruppo di ricerca HuggingFace ed al modello DistilBERT in particolare. Questo modello utilizza il paradigma Teacher-Student (Insegnante-Studente) per trasmettere la capacità di generalizzazione tra due modelli (verrà spiegato in maggior dettaglio nella sezione 5.2.1) e promette di preservare il 97% della conoscenza di BERT, usato come Teacher dal gruppo di ricerca, ma riducendo considerevolmente le dimensioni ed aumentando in velocità di training ed inferenza.

Infine verranno presentati i risultati ottenuti e spiegati i miglioramenti che, seppur modesti, risultano in una conservazione della conoscenza del 94% rispetto al Teacher utilizzato. A livello di modelli sia per lo Student che per il Teacher verranno proposti livelli di question answering modificati che vedremo nel relativo capitolo (6). Inoltre saranno comparati anche risultati ottenuti su OLP in diverse configurazioni con lo scopo di dimostrare come un modello come BERT possa essere messo in difficoltà quando non si lavora con un ambiente che rispetta certi vincoli.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In the few last years, the Natural Language Processing (NLP) field has seen a great expansion, also on large scale distribution. With the last technologies, understanding and processing human language became an important part in research, making computer and other devices more human friendly.

Nowadays, we all have already used, at least once, systems like **Amazon ALEXA**, **Google Home** or **Apple Siri**. This is only a part of this field, which is composed by many other sub-fields like speech recognition, text generation and speech generation, text or natural language understanding. NLP plays an important role also as a baseline to create support devices for people with diseases, starting from a simple screen reader to a more complex kind of information retrieval.

We found the purpose of our studies in **text comprehension** and focused on the specific part of question answering by using the BERT model, which is State-of-the-Art at the moment for many NLP tasks. In the next section we will see an overview on what Natural Language Processing mean.

1.1 Natural Language Processing

In short, the aim of this artificial intelligence field is to intermediate communication between humans and machines by using the natural language. Let us see some definitions (you can find them online) which explain the concept with better words.

Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language. The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable. Most NLP techniques rely on machine learning to derive meaning from human languages.[1]

Natural Language Processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.[2]

Natural Language Processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language. NLP draws from many disciplines, including computer science and computational linguistics, in its pursuit to fill the gap between human communication and computer understanding.[3]

NLP can be divided in four macroareas:

- Syntax

[1]`https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32`

[2]`https://en.wikipedia.org/wiki/Natural_language_processing`

[3]`https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html`

- Semantics
- Discourse
- Speech

Of course, all of these areas have many subtasks like speech recognition, part-of-speech tagging, automatic summarization etc.

Ambiguity is one of NLP keywords. The ambiguity of natural languages makes all of the NLP tasks really challenging. We will focus on Question answering, which is part of the *Semantics* macroarea.

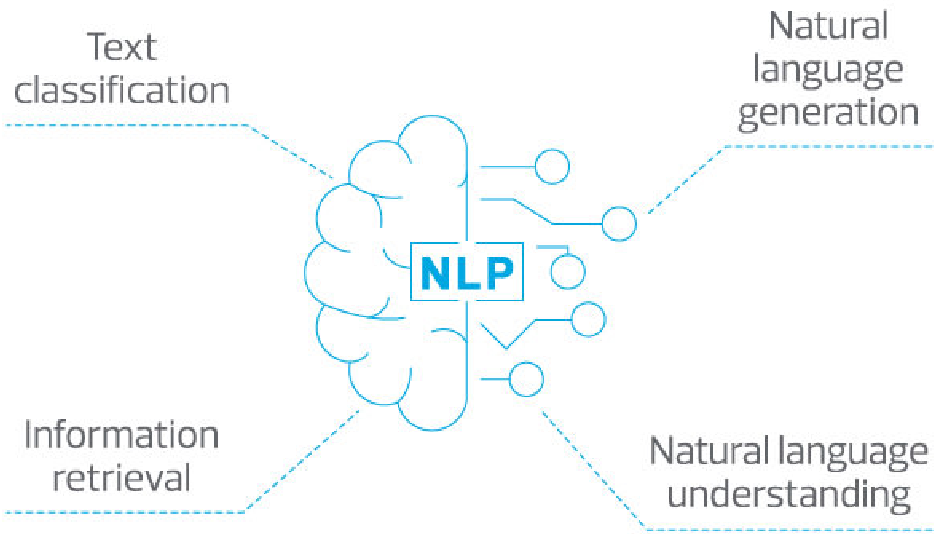


Figure 1.1: Natural Language Processing[1]

Chapter 2

Neural Networks

2.1 Artificial Neural Networks

An Artificial Neural Network, in the Machine Learning field, is an artificial model composed of connected **neurons** which should reproduce a biological neural network. In other words, the Artificial Neural Network idea is directly inspired by the human brain structure. These networks are represented by algorithms and they recognize numerical patterns, so we need to convert sensory data information into numerical representations before feeding them to the neural network.

Like signals transmitted through synapses, real numbers are transmitted among neurons through the connection of the artificial network. We will see the **neuron** and its possible activation functions more in details in the next sections.

2.1.1 Neuron

The neuron is the fundamental part of an Artificial Neural Network. This component receives a signal (a real number) input, makes some computation and sends its output to the other neurons. A neuron can receive more than one input at the same time; it processes all these signals to produce an unique output. Normally it sums all the inputs $x_i w_i$, where x_i is produced by the i-th neuron and w_i is the weight associated to the connection, and adds a bias (if it exists) before feeding the result to the activation function.

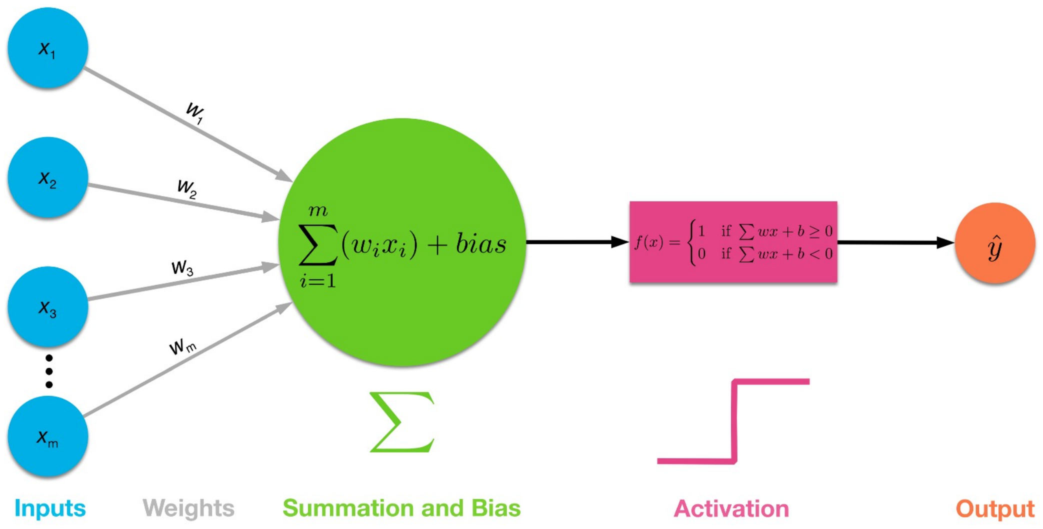

x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
$\sum$
Inputs
Weights
Summation and Bias
Activation
Output

Figure 2.1: Artificial Neron

2.1.2 Activation Functions

An Activation Function represents the final elaboration step of the neuron and it is performed before sending the output to other neurons.

Nowadays, the most popular function is the Rectified Linear Unit, also called ReLU [2], which allows a faster convergence. Other examples of activation functions are the sigmoid and the tanh, defined as:

$$sigmoid(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

$$Tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \tag{2.2}$$

while the ReLU is simply defined as:

$$ReLU(x) = max(0, x) \tag{2.3}$$

ReLU is fast to compute since its value is equal to the identity and 0 for all the negative x, so it is faster both during training and at inference time.

In the next chapters, while talking about BERT, we will see the GELU [14] activation function used inside BERT by the Google AI research group. This function is defined as $xP(X \leq x) = x\phi(x)$ which can be approximated to:

$$GELU(x) = 0.5x[1 + Tanh(\sqrt{\frac{2}{\pi}}[x + 0.044715x^3])] \tag{2.4}$$

This variant should preserve neurons from dying, since the negative values try to mitigate this case without eliminating it totally.

2.2 Type of ANN

2.2.1 Feed-Forward Neural Networks

The Feed-Forward Neural Network (FNN) [4] is an Artificial Neural Networks that does not present loops. It is the first simplest model of ANN in which the information goes only forward, starting from the input nodes to the output ones and passing though the so called **Hidden Layers**.

The most simple example of a Feed-Forward network is the **Simple Perceptron**, composed only of the input and the output layers. Its neurons are fully connected between the two layers while the **Multi-Layer Perceptron** is composed of at least two layers of neurons without considering the input one instead. This generalization allows neural networks to represent and approximate complex non-linear functions.

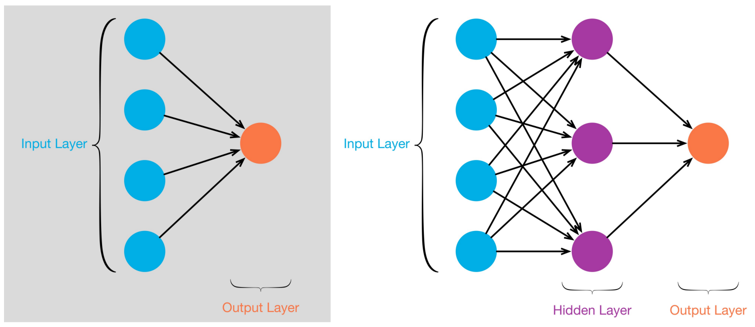


Figure 2.2: Single and Multilayer Perceptron

2.2.2 Convolutional Neural Networks

The Convolutional Neural Network (CNN) [24] is a special case of Multi-Layer Perceptron inspired by biological processes. This tries to represent the brain visual cortex. A CNN is composed of one or more *Convolutional Layer* and could also have some *pooling* layers and finally a linear fully connected network.

A **Convolutional Layer** extracts the features from an image. It acts similarly to a scanner moving a particularly small matrix, called kernel, over the entire image. Each convolutional layer extracts different kinds of features because these matrices act like filters. A convolution is a kind of parameter sharing, since each filter extracts the presence or the absence of a feature in an image, which is a function of not just one pixel but also of its surrounding neighbor pixels.

A **Pooling Layer** is normally applied after a series of Convolutional Layers in order to downsample the input by reducing its dimension but preserving the features' information.

The last level is composed of one or more **Fully Connected Layers** and its result is a dot product between the output of the previous layer and the weights' matrix, which has to be trained.

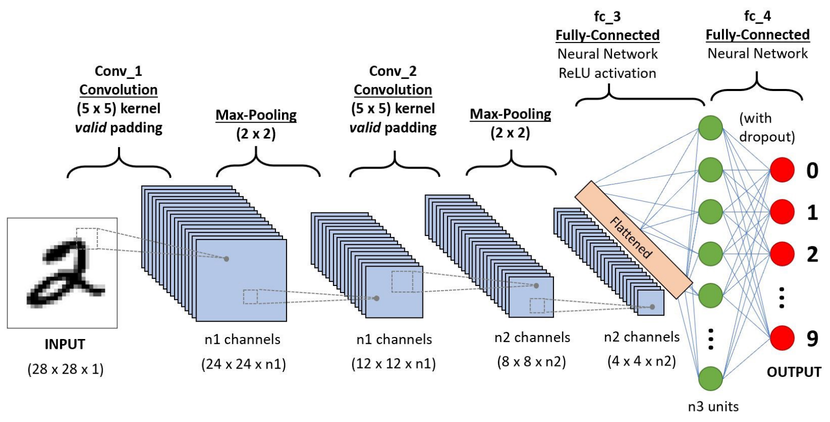


Figure 2.3: Representation of a Convolutional Neural Network

2.2.3 Recurrent Neural Networks

The keyword of the Recurrent Neural Network (RNN) [35] is **memory**. What does it mean? It means that, unlike a basic Feed-Forward Network, RNN remember things not only from the current training, but also maintain some *context* from previous inputs, the so called **Hidden State Vectors**. For example, the same input can produce different outputs, it depends on the previous inputs. In other words, a permutation of the input sequence normally leads to producing different outputs.

This kind of network is particularly suitable for tasks that need the help of a context, such as speech recognition and other Natural Language Processing tasks.

Parameter Sharing

RNN shares parameters across inputs. When this kind of network do not share them, it is only a normal FNN where each input has its own weights. The most commonly cells used in RNN are GRU [8] and the Long Short-Term Memory (LSTM) [37].

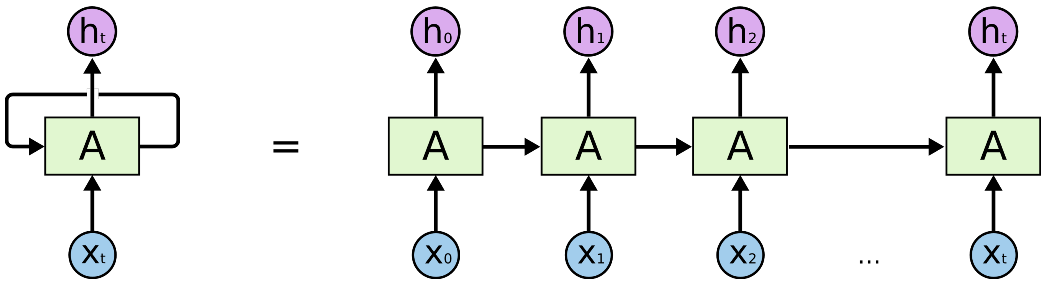


Figure 2.4: Simple representation of a Recurrent Neural Network

Encoder-Decoder Sequence to Sequence RNN

For translation services, this network is composed of two Recurrent Neural Network, the **Encoder** and the **Decoder**. The first one produces the context output, the encoder vector that is fed to the decoder part which translates it to a sequence of outputs.

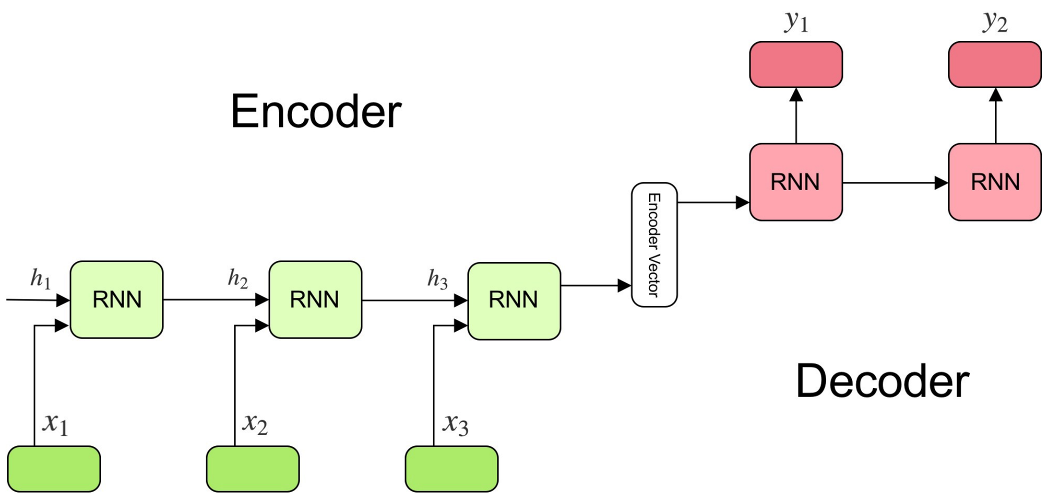


Figure 2.5: Encoder-Decoder with Recurrent Neural Network

2.3 Training

2.3.1 Hebb's Rule

"Cells that fire togheter, wire together", this is a summarize of the Donald Hebb's postulate. It is not completely correct because the original postulate says *"A cell A takes* ***part*** *in the activation of the cell B"*[15], where A and B are two connected neurons. The weights are updated during the training phase at every training example, for a graphic representation have a look to the Picture 2.1.

The Hebb's rule is obsolete and does not accurately describe the behaviour of a human brain. In fact, it assumes that a connection has to be strengthened independently whether the result is good or not.

2.3.2 Backpropagation

Nowadays, *Backpropagation* [32] is used, which is a supervised learning technique that consists in minimizing the loss function by calculating the **gradient** (descent in case of minimizing, ascent instead).

The gradient is the multi-variable generalization of a derivative of a function and determines how a weight value has to change and whether the corresponding connection has to be reinforced or not. The two main reasons to calculate the gradient are the following ones: the derivative shows the direction of the cost function and also how much the weight needs to change to minimize that function. After an input goes through a neural network, we calculate the gradients and the new weight, which are pushed back in the neural network to update all the weights inside it.

Although the gradient descent gives good results when changing weights, it is really slow and this reflects negatively on the training time, so we need to use some optimizers such as **Stochastic Gradient Descent (sgd)** [42] and **Adam** [19]

2.4 Word Embedding

The Word Embedding is a kind of document representation in the Natural Language Process field. It could be seen as a learning technique in which the words are translated in their real number vector representation, so they can conserve and give information about the context and the semantic for each word in a document. We need this representation because our aim is to capture word dependencies to reinforce the concept of context and have better information about word relations in a text.

One of the most famous technique which implements this idea is **word2vec** [26]. It uses Neural Networks and either Common Bag of Words (CBOW) or Skip-Gram [27]. The first one predict the word based on the context while the second one predicts the surrounding words given the current word.

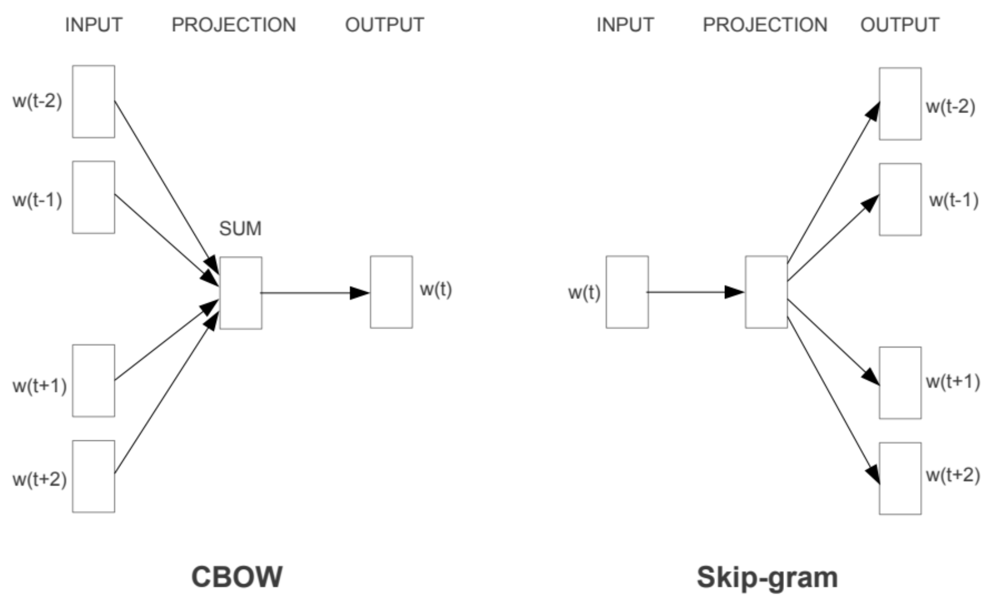


Figure 2.6: Comparison between CBOW and Skip-Gram

2.5 Dropout

The overfitting problem in Neural Networks consists of a model unable to generalize, for example because it can become too specific about the training set. This means that the model will make good predictions on data seen during the training, but it will have bad performances when applying what it learnt to unknown data.

This problem is mitigated though normalization techniques such as Dropout [36]. It basically ignores randomly some connections by "dropping" them temporarily and by avoiding to correct mistakes from previous layers. This situation is called co-adaptation and could lead to overfitting because the network does not generalize on unknown data.

Chapter 3

Transformers

3.1 Transformers Architecture

Transformers is the actual State-of-the-Art in NLP introduced by the “Attention is all you need” paper [39]. A transformer is usually composed of two parts: the Encoder and the Decoder. Both of them have an Attention mechanism inside. Let us show in a more detailed way, in the next sections, the Encoder and the Decoder.

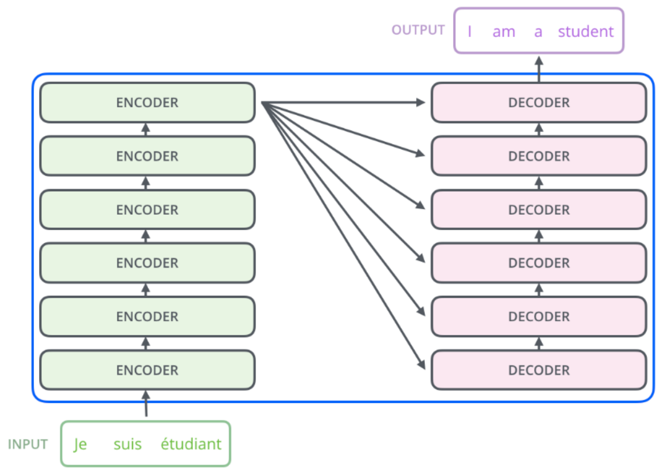


Figure 3.1: Transformers high-level structure

3.1.1 Encoder

If we explode the Encoder and look inside it we will actually find many Encoders, going deeply in each of these “layers”, which have the same structure. We will find a Multi-Head Attention layer with a normalization layer above and a Feed-Forward Network (a two layers network with ReLU activation function in between them) with another step of normalization on the top. The input goes through all the Encoder layers and at the end the final output is passed to each Decoder layer at the same time. We prefer to explain the Self-Attention and the Normalization in the 3.1.3 section.

3.1.2 Decoder

The Decoder part apparently uses the same structure as the Encoder part, but having a look inside a Decoder layer, we can notice that there is an additional layer, the so called *Masked Multi-Head Attention* with normalization. Then, we will find an Encoder-Decoder Attention layer which “connects” the Encoder part to the Decoder one. In fact, this layer receives directly the encoder output, then the output of this layer is normalized, fed to the Feed-Forward Network and normalized before passing the final output to the next Decoder. In the end, the final decoder output goes through a final linear layer with a softmax on the top.

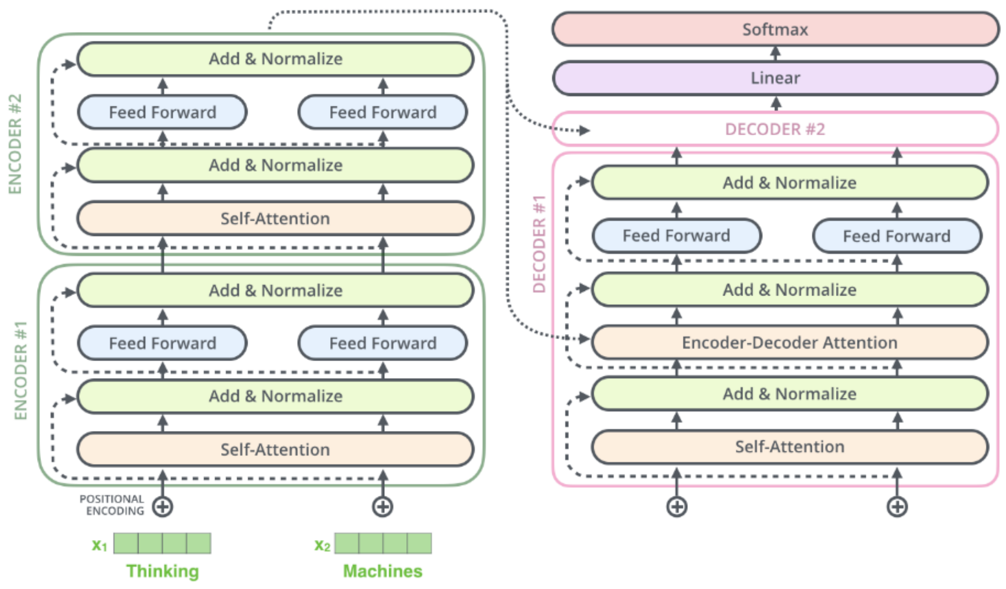


Figure 3.2: Encoder-Decoder connection and Attention

3.1.3 Attention

Definition: *Self-attention, sometimes called intra-attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.* [39]
Let us show *Attention* formulas before explaining each element in detail in the next paragraphs.

$$A = \frac{QK^T}{\sqrt{d_k}} \tag{3.1}$$

$$Attention(Q, K, V) = softmax(A)V \tag{3.2}$$

Self-Attention

Self-Attention could substitute the LSTM [37] network with a new method which takes the relation between the current word and all the other words in the text into account.

Now we will see how to compute this attention to understand better what the concept of word VS words means. It is necessary to calculate three important elements, which are called embeddings, for each word. Those elements are **Q, K, V**, *Query, Keys* and *Values* respectively; to produce these embeddings we need three different matrices which are learnt at training time on loss back-propagated. In brief, each word embedding is multiplied for each matrix to obtain the corresponding embedding. Assuming we are calculating the embeddings for the word x^1, the first step is to multiply $x^1 emb$ for each matrices:

$$x^1 emb * W_Q = Q_1 \tag{3.3}$$

$$x^1 emb * W_K = K_1 \tag{3.4}$$

$$x^1 emb * W_V = V_1 \tag{3.5}$$

Here is an example to understand better what each of these embeddings represents: assuming that a word x^1 wants to know its value with respect to another word, it has the possibility to query (our **Q**) the other word x^2, which will provide an answer (the **K**). The score is a simple dot product between **Q** and **K**, $Q \cdot K$. This will be performed for each word, then a *softmax* function

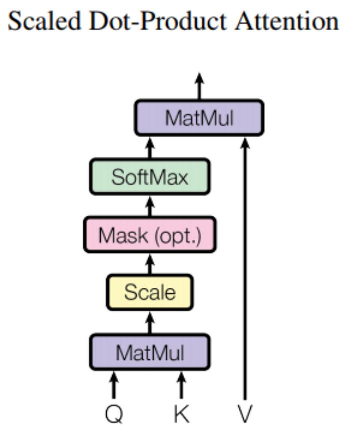


Figure 3.3: Scaled Dot-Product, from equations: 3.1 and 3.2 [39]

is applied to all these scores to ensure a relative difference between scores.

This step is performed by every word against all other words (the word VS words we named above). The scores are now used by the word to obtain a new value of itself w.r.t. the other words, in our case x^2; depending on the score, the corresponding **V** could be reduced or reinforced. The final embedding, or better, the new word embedding, is given by summing up all the Value embeddings, as shown in the figure 3.3.

Multi-Head Attention

Since we obtain many W_Q, W_K, W_V after the training, for each matrices set and each word, we need to calculate many V_1'(e.g. for the first word). All those embeddings have to be concatenated and thus multiplied with a (learned) **Z** matrix to produce an unique embedding for the word x_1'. The

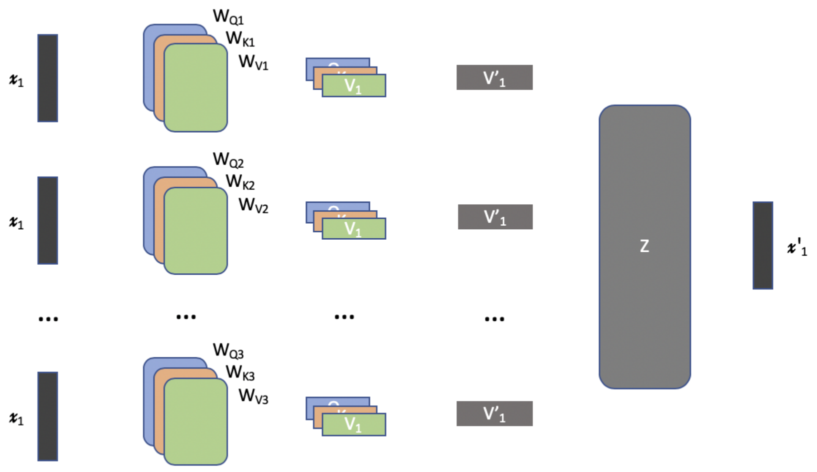


Figure 3.4: Multi-Head Attention [10]

multi-head idea is to obtain a final embedding which takes into consideration diverse contexts at the same time. This result could be reached by initializing all the **Q, K, V** matrices randomly and by training them with loss backpropagation.

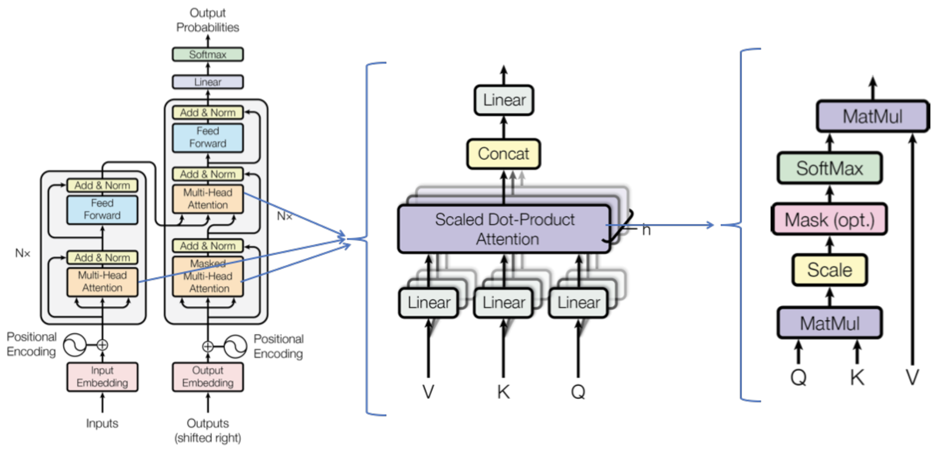


Figure 3.5: Representation of where Multi-head Attention is located inside the transformers structure [40]

3.1.4 Layer Normalization

There are two main reasons to use **Layer Normalization**[3] instead of **Batch Normalization**. The first is that with a batch size of 1 the variance is zero and in this case the batch normalization can't be applied; for this reason, also small values of batch size produce noise that has a negative impact on the training. The second reason is Recurrent Neural Networks (RNNs), the model become more complicated since the recurrent activations of each time-step will have different statistics and we are forced to store statistics each step-time during training [18]. **Layer Normalization** normalizes the input across the features rather than across the batch dimension, as we can see in the following formulas:

Batch Normalization	Layer Normalization
$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_{ij}$	$\mu_j = \frac{1}{m}\sum_{j=1}^{m} x_{ij}$
$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_{ij}-\mu_j)^2$	$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m}(x_{ij}-\mu_i)^2$
$\dot{x} = \frac{x_{ij}-\mu_j}{\sqrt{\sigma_j^2+\varepsilon}}$	$\dot{x} = \frac{x_{ij}-\mu_i}{\sqrt{\sigma_i^2+\varepsilon}}$

Table 3.1: Batch Normalization vs Layer Normalization Formulas

Compute statistics across the features means that they are independent from other examples. A graphical explanation could help to understand what “across the features” means (Figure 3.6).

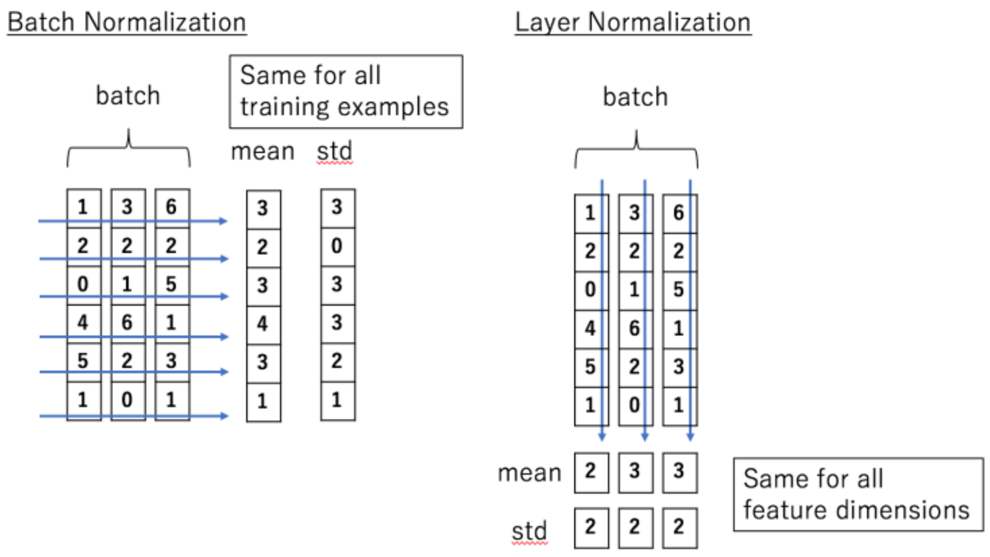


Figure 3.6: Batch Normalization vs Layer Normalization[18]

3.2 BERT

BERT model is considered a State-of-the-Art in many NLP tasks like Question Answering (QA) or Natural Language Inference (MNLI). It uses a bidirectional training strategy (Figure 3.7).

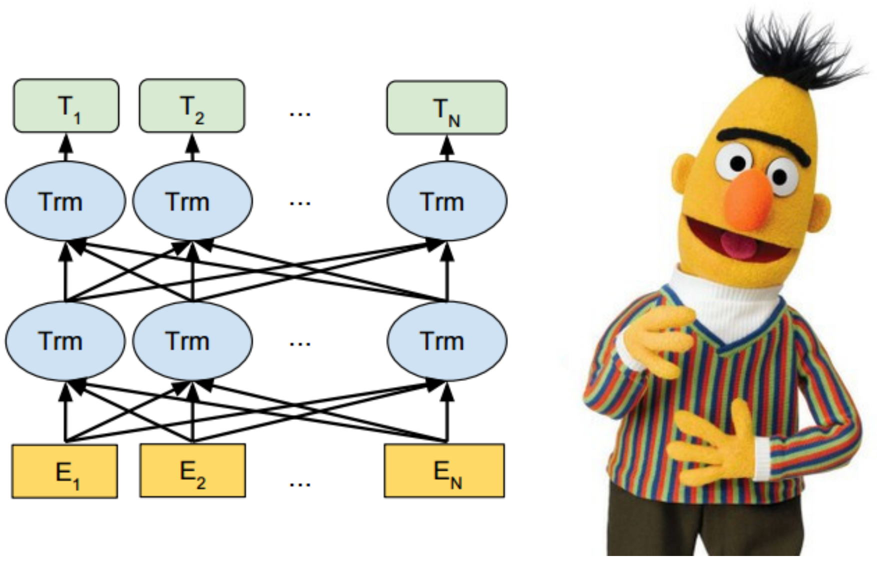


Figure 3.7: BERT [9]

BERT stands for **B**idirectional **E**ncoder **R**epresentation from **T**ranformers. Released in the late 2019, it does not need to have the decoding part since its aim is to generate language. This bidirectional mechanism allows BERT to learn the context of a single word from the two words that surround it (left and right). Later, the model reads the entire sentence at once[17]. BERT was released in two different architectures, that is $BERT_{BASE}$ and $BERT_{LARGE}$. The smaller BERT presents 12 layers (transformers blocks), hidden size of 768, 12 self-attention heads and a final number of parameters of 110 million, while $BERT_{LARGE}$ is a huge model, 24 layers, hidden size of 1024 and 16 self-attention heads, resulting in 340 million of parameters.

$Model$	$NumLayers$	$HiddenSize$	$Self-AttentionHeads$	$NumParameters$
$BERT_{BASE}$	12	768	12	$110M$
$BERT_{LARGE}$	24	1024	16	$340M$

Table 3.2: $BERT_{BASE}$ and $BERT_{LARGE}$ Dimensions

We will see the training strategies in the next two sections, which are divided in two phases, both about the BERT model. The training of BERT is divided in two, the pre-training, which is the biggest part and takes a considerable time and uses unlabled data, and the fine-tuning part, which prepares BERT for a specific task.

3.3 Transfer Learning

With the term **Transfer Learning** we refer to the technology which allows us to use the knowledge, acquired by training a model for a task, in order to solve another related task. In this way is is possible to recycle a huge amount of time and computational effort. Its aim can be explained, in other words, as a different use of a model output. Let us take a practical example: we do **Transfer Learning** when we use BERT prediction to solve the **Question Answering** task by using an additional level to manipulate that output. This was used for a long time with the Convolutional Neural Networks and it is divided in two phases: the first one consists in getting a pretrained-model which has been trained for long and, normally, on a big set of GPUs, and a second one, where we use fine-tuning by using a new more specific and smaller dataset for the task we want to solve or, in case of the new dataset being very large by using the pre-trained model weights to initialize the new model. This concept has never been used in Natural Language Processing before BERT release.

3.3.1 Pre-Training

Before starting with the pre-training, it is necessary to briefly introduce the elements that help the model in this task. These actions are performed before entering the model:

- [**CLS**] token: begin of the first sentence.
- [**SEP**] token: end of each sentence.
- The Sentence embedding is added to each token to recognize which sentence it comes from.
- The Positional embeddings add the information about the position of the token is in the sentence.

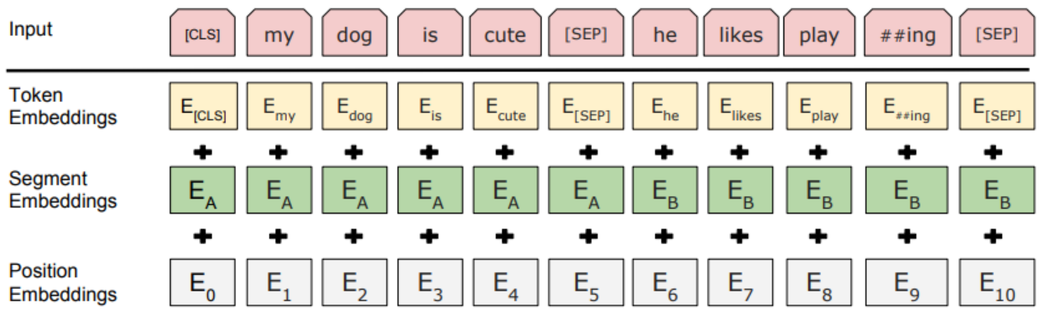

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 3.8: Representation of diverse embeddings in NSP training [9]

As it is shown in the picture 3.8, BERT takes in input a concatenation of two segments composed of tokens. This concatenation is a single output sequence with special tokens delimiting the two segments. The sequence length is controlled by a parameter. So, assuming N is the first segment's length and M is the second segment's length, if T is the maximun sequence length that BERT can evaluate, then it must be that the sum of N and M is less than T: $N + M < T$. The training was performed by using GELU [14] activation function instead of the normal ReLU and the loss comes form the sum of the mean masked LM likelihood and the mean next sentence prediction likelihood.

Masked LM (MLM)

MLM is a bidirectional approach which is used instead of the classic left-to-right. This method starts setting the 15% of the total words as **"possible replacement"** in each sequence by using the following criterion: 80% of these words are substituted by the token [**MASK**], 10% are kept unchanged and the remaining 10% are changed with randomly chosen words. The model should attempt to predict the original words by considering the context from the words that are not marked as **"possible replacement"**. Since this approach considers only 15% of the words, it converges slower, but still with better results. The model needs an additional classification layer on the top of the encoder. The transformer has to keep all of the contextual representation of every input token because it does not know which word could be replaced or which one it has to predict. The model has to be trained for a long time because MLM converges slower than normal left-to-right models.

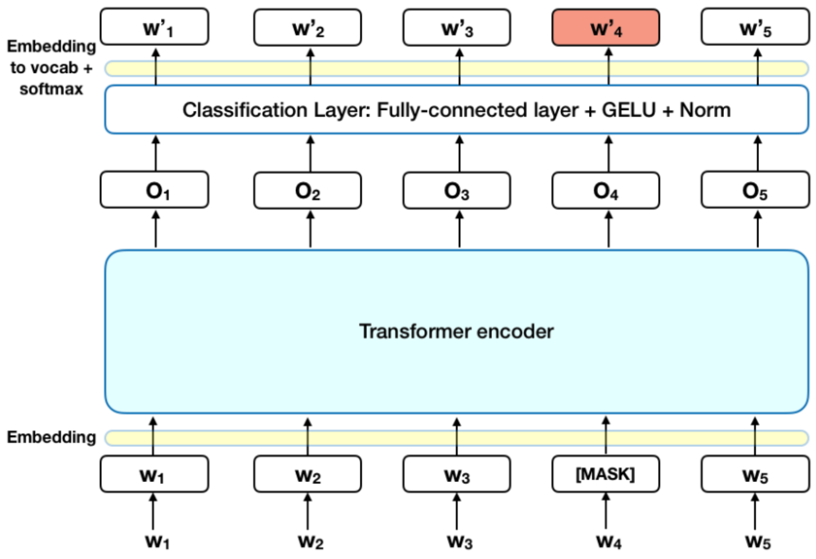

W'1
W'2
W'3
W'4
W'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
W1
W2
W3
[MASK]
W5
W1
W2
W3
W4
W5

Figure 3.9: MLM training mechanism [17]

Next Sentence Prediction (NSP)

This sub-task is really useful for tasks like question answering. The system receives a pair of sentences in input and learns whether the second sentence is the one that follows the first sentence in the original text. The training set is 50% balanced. In other words, one half of the input is composed of real subsequent sentences, while the other half is composed of disconnected sentences, which are randomly chosen.
Steps of prediction:

1. The whole input sequence goes through the transformer model.
2. A classification layer transforms the output of [**CLS**] token in a 2x1 vector.
3. The probability of *isTheNextSentence* is calculated via softmax.

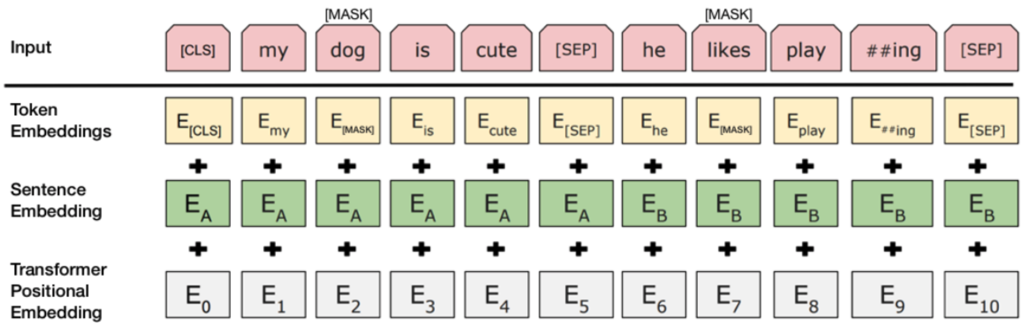

Input
[CLS]
my
[MASK]
dog
is
cute
[SEP]
he
[MASK]
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} $E_{[MASK]}$ E_{is} E_{cute} $E_{[SEP]}$ E_{he} $E_{[MASK]}$ E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Sentence Embedding
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Transformer Positional Embedding
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 3.10: Representation of diverse embeddings in NSP training [17]

MLM and NSP are trained together because the goal, when training BERT, is to minimize the combined loss function of both strategies.

3.3.2 Fine-Tuning

BERT is a really flexible model because it is sufficient to add a simple additional layer on the top of it.

Depending on the task we are dealing with, we have to choose the correct type of the last layer. For every up-level task we can directly use the BERT part, which is already trained, and train only the layer for the specific task by using BERT weight. In this way it is not necessary to train the whole model every time. Here are some of the tasks we do by using a fine-tuned BERT:

- Classification Tasks
- Question answering
- Named Entity Recognition

Question answering is the task in which we are interested the most. It consist in "marking" a span in the sequence as an answer for a question posed in natural language. BERT can be trained to learn two vectors that mark the beginning and the end of the answer. Different datasets were created and each of them could be used to fine-tune BERT, e.g. TriviaQA and SQuAD. SQuAD is the baseline used in our work and we will explain its characteristic deeply in the sections 4.2.1 and 4.2.2.

BERT could be treated as a black box, which could be used without knowing how it works inside. What we need to study is its output and how manipulate it to solve the specific task.

SQuAD v2 Representation

Questions which do not have an answer are treated as questions which have one, but with an answer, span with start and end positions at [**CLS**] token. This means that this value is normally zero. Thus, the probability space had to be extended to include the position of the [**CLS**] token. When we predict, we compare the score of no-answer span, s_{null}, to the best non-null's score span, $s_{\hat{ij}}$. We predict a non-null answer when $s_{\hat{ij}} \geq s_{null} + \tau$[9], where τ is chosen to maximize the F1 score, we will explain it better and show the formulas also for Exact Match in the 4.3 section talking about metrics.

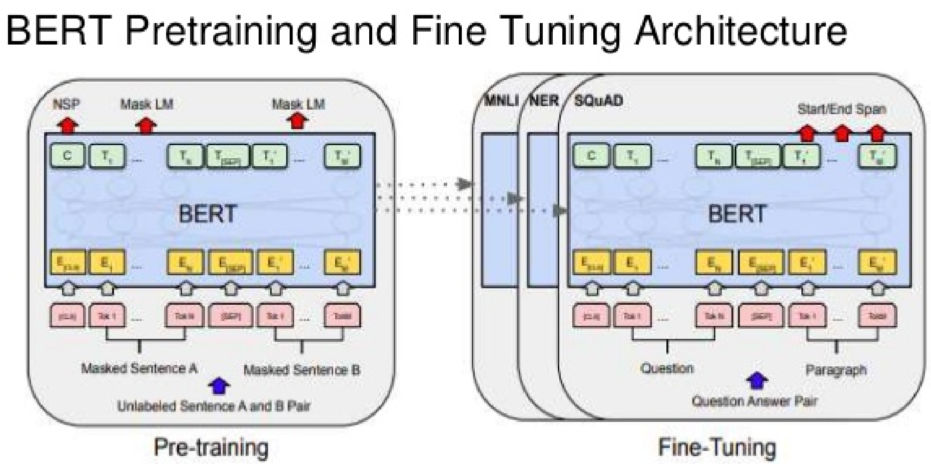


Figure 3.11: On the right is represented BERT as model and on the left the highest levels which provide the task specific part [9]

3.4 Derived Models

The next few works are a very small part of the set of works on BERT model. This underlines the importance of BERT in NLP field, since a lot of people and teams demonstrate their interest in this model.

Researchers, when BERT came out, immediately noticed the huge dimension of the model and, consequently, the long time of training it requires, then its training efficiency with the correct parameters. These characteristics hinder BERT from being used on edge devices such as smartphones. The following models all derive from BERT and tried to solve these problems.

3.4.1 RoBERTa

RoBERTa stands for **R**obustly **O**ptimized **BERT** pre-training **A**pproach [25]. As it has been already said, one of the key points, when approaching to a model like BERT, is to have a well trained model. This research group is composed of people from the University of Washington and the Facebook AI research group. This work was born from the idea that BERT was significantly **undertrained**. They started doing experiments on the $BERT_{BASE}$ configuration as a first try for the diverse train strategies.

Training Strategies

- Static Masking & Dynamic Masking: the original version of BERT performs masking only once, before feeding the model. The training data were augmented by 10 times by masking sequence in 10 different ways to avoid the problem of Static approach. The RoBERTa group invented a new system, **Dynamic Masking**, that generates masking patterns every time, avoiding data duplication and occurrence of the same masking patterns several times during training.

- Model Input Format and NSP: NSP resulted actually as an irrelevant task, so they removed it in favour of blocks of texts. **FULL-SENTENCES** input can cross document boundaries and add an additional token, which indicates the document ending. The input sequence is shorter than 512 token. **DOC-SENTENCES** is similar to **FULL-SENTENCES**, except for the possibility of document boundaries crossing.

- Training with large batches: the batch size seems to have a consistent influence in term of speed and performance. BERT was trained for 1 million steps with batch size of 256, which is equivalent to training BERT for 125 thousand steps with batch size of 2 thousand or 31 thousand steps with batch size 8 thousand. Training with larger batch size improves **perplexity** (how well a probability model predicts test data. In the context of Natural Language Processing, perplexity is one way to evaluate language models. Since it is an exponential of the entropy, the smaller its value is, the better).

- Text Encoding: Byte Pair Encoding (BPE) hybrid between words and character-level, BPE uses bytes instead of characters.

In the end, the new training system was built by considering the results obtained in the intermediate step. The final results are composed of a **Dynamic Masking** approach, **FULL-SENTECES** without NSP task, **larger mini-batches** and **larger byte-level BPE**. After the latest decision, they started to use a bigger version of BERT, $BERT_{LARGE}$. Firstly, $BERT_{LARGE}$ was trained with RoBERTa settings for 100 thousand steps with BOOKCORPUS [44] plus WIKIPEDIA English, obtaining a first improvement with respect to the results published in the BERT paper. Then, they performed three more trainings for 100, 300 and 500 thousand steps and combined the last dataset with three more, CC-News [7], OPEN WEB TEXT [11] and STORIES [38], for a total dimension of 160GB. We propose here a summary of the most important results on SQuAD v1.1 and v2 published in the paper [25]:

$Changes$	$DataDimension$	$BatchSize$	$Steps$	$SQuADv1.1$	$SQuADv2$
$BOOKS+WIKI$	$16GB$	$8K$	$100K$	93.6	87.3
$+OTHERDATASETS$	$160GB$	$8K$	$100K$	94.0	87.7
$+TRAINEDMORE$	$160GB$	$8K$	$300K$	94.4	88.7
$+TRAINEDEVENMORE$	$160GB$	$8K$	$500K$	94.6	89.4
$BERT_{LARGE}$	$13GB$	$256K$	$1M$	90.9	81.8

Table 3.3: Comparison between RoBERTa and BERT results on QA task using SQuAD Dataset (F1 score)

3.4.2 ALBERT

Differently, ALBERT's goal is to reduce the dimension of the model and give a speedup in terms of training time. Two main intuitions led to the final "smaller" model: **Factorized embedding parametrization** and **Cross-layer parameters sharing**.

ALBERT promises to be 1.7 times faster than the original BERT and to have 18 times less parameters, which could be seen also as a form of regularization that helps with generalization.

Architectural Choices

As we have said above, BERT uses transformers encoders with GELU which is a non linearity activation function. We will see here ALBERT's contribution:

- **Factorized embedding parametrization**: E is the wordPiece (context-independent learning) embedding size, which is directly related to H, the hidden size (hidden layer context-depending learning) and they take normally the same value. It is possible to gain a more efficient use of the total model parameters by separating these two values if $H >> E$. Since the vocabulary is usually very large and the dimension of the wordPiece[41] embeddings is given by VxE, the increase of H (and E) results in a billion of useless model parameters.
  With ALBERT, the authors divide embedding parameters in two matrices; in this way, the number of parameters decays consistently from $O(VxH)$ to $O(VxE + ExH)$ when $H >> E$. They choose to have the same E for all word pieces because these are much more evenly distributed across the documents compared to whole-word embedding [23].

- **Cross-layer parameters sharing**: This would be also an improvement in terms of parameter efficiency. Although there are many ways of sharing, in this work it was decided to use all of the parameters sharing across layers. After many experiments, this method led to better results, as shown in tables 4 and 5 in the paper [23].

- **Inter sentence coherence loss (SOP)**: Since NSP was a really trivial task and did not have a substantial contribution during the training phase, it was replaced by Sentence Order Prediction loss. SOP uses consecutive sentences (just like BERT) for the positive samples and swaps its sequence for the negative examples instead of choosing random sentences from the text.

Model and experimental Setup

In the following table (3.4) all the configuration of ALBERT and BERT are summarized to have a better idea about what changes between the two models, especially in terms of number of parameters:

Model	*Parameters*	*Layers*	*Hidden*	*Embedding*	*ParametersSharing*
$BERT_{BASE}$	$108M$	12	768	768	*False*
$BERT_{LARGE}$	$334M$	24	1024	1024	*False*
$BERT_{XLARGE}$	$1270M$	24	2048	2048	*False*
$ALBERT_{BASE}$	$12M$	12	768	128	*True*
$ALBERT_{LARGE}$	$18M$	24	1024	128	*True*
$ALBERT_{XLARGE}$	$60M$	24	2048	128	*True*
$ALBERT_{XXLARGE}$	$235M$	12	4096	128	*True*

Table 3.4: Comparison between BERT and ALBERT dimensions and number of paramenters

The same dataset used with BERT (BOOK CORPUS + English Wikipedia) was used for the pre-training of the model, by formatting the input in the following way:

"[**CLS**] $sentence_1$ [**SEP**] $sentence_2$ [**SEP**]".

The system was set to a maximum input length of 512 and a probability of 10% of input shorter than 512. The vocabulary dimension is of 30000 like in BERT but it was tokenized by using **SentencePiece**[22] like in XLNet[43]. In addition, the masked inputs were generated by using n-gram with n of maximum 3 and for a 125 thousand steps. To conclude the ALBERT chapter, we would like to present its results on Question Answering task obtained by using SQuAD v1.1 and v2 in comparison with BERT (Table 3.5):

$Model$	$SQuADv1.1$	$SQuADv2$
$BERT_{BASE}$	90.4/83.2	80.4/77.6
$BERT_{LARGE}$	92.2/85.5	85.0/82.2
$BERT_{XLARGE}$	86.4/78.1	75.5/72.6
$ALBERT_{BASE}$	89.3/82.3	80.0/77.1
$ALBERT_{LARGE}$	90.6/83.9	82.3/79.4
$ALBERT_{XLARGE}$	92.5/86.1	86.1/83.1
$ALBERT_{XXLARGE}$	94.1/88.3	88.1/85.1

Table 3.5: Comparison between BERT and ALBERT results on QA task using SQuAD Dataset (F1/EM respectively)

3.4.3 DistilBERT

DistilBERT approaches the dimension and the speed problems with another technique. The main idea of the study is to create a considering smaller model with the same structure of the bigger one. For example, in DistilBERT the pooler and the token-type embeddings were removed. In addition, it has less numbers of levels. We will explore this model better in sec: 5.3.

For now let us anticipate in a really brief description what they did. They "Transferred" knowledge from the bigger model (BERT) to the smaller one (DistilBERT) by using distillation, considerably reducing the training time as well as the final dimension of DistilBERT.

Chapter 4

Question Answering

4.1 Question Answering Problem

Question Answering is that specific field of Natural Language Processing (NLP) which tries to reproduce human behaviour when answering questions posed by humans. In 1960s, the first approaches were called BASEBALL and LUNAR. Both of them had a restricted domain to search answers. After that it was necessary for the research to focus more on improving these techniques in information-retrieval.

One thing which is important to notice is that both of them use a closed-domain. The research is limited to a specific topic, so the questions were given on that context. Recent discovery in NLP (Natural Language Processing) permits to work on opened-domain datasets. In brief, closed-domain datasets and opened-domain ones differ in context, since opened-domain datasets do not have a specific topic. We can divide the question answering problem in four steps, which could be considered as the QA architecture.

The four steps are:

1. Question Analysis

2. Document Retrieval

3. Answer Extraction

4. Answer Evaluation

The process flow is shown in the next picture (4.1).

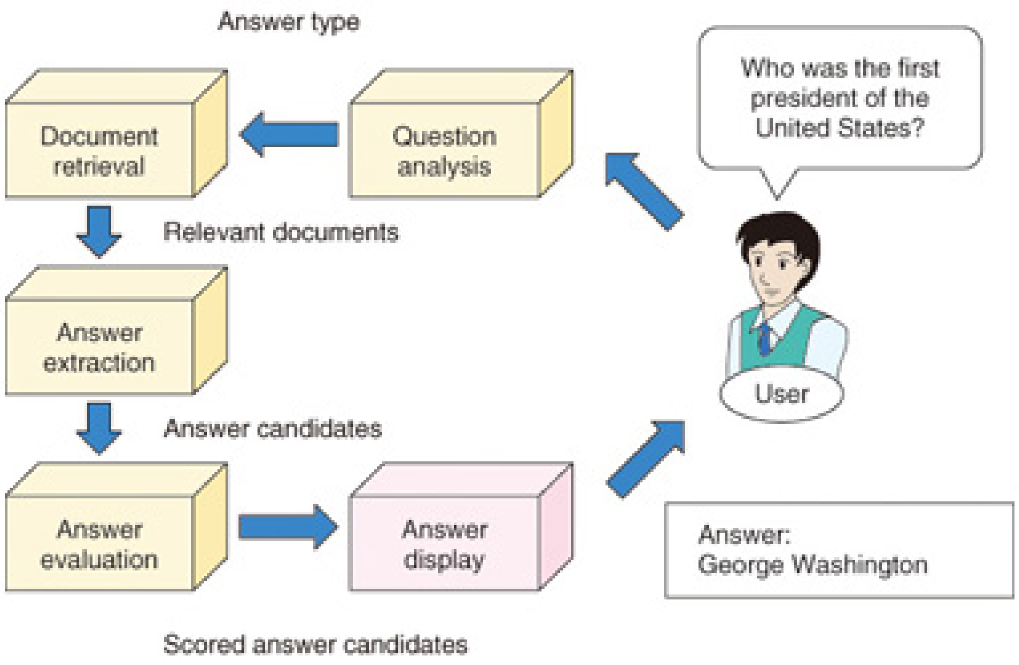


Figure 4.1: Question Answering steps [16]

At the moment, with BERT, it is relatively simple to find an answer to well formulated questions that have a specific right answer. In this task the newest models reached human performances, while understanding if there is an answer in the text, and in case finding it, is still an interesting challenge. A more complex task is to find questions through text comprehension. In other words, it is really difficult to find an answer that needs to capture some text semantics, for example in the case of the answer being distributed in different parts of the text. Anyway, it is possible, thanks to the last research, to consider also open-ended and multi-answer questions.

4.2 Datasets

In all the machine/deep learning applications it is necessary to have a good dataset, from which a model can learn how to solve the problem. The Stanford Question Answering Dataset (SQuAD) [31] was used as a baseline.

The SQuAD dataset is the result of crowdworkers work. It consists in questions posed by humans and it is available in two versions, v1.1 [30] and v2 [29]. The second version adds the possibility to have unanswerable questions. This means that some questions do not have an answer in the article under consideration.

The second dataset, the OLP dataset ([6], [5]), is an experimental one. It is still work in progress. The aim of this dataset is to be useful for other text comprehension tasks.

4.2.1 SQuAD v1.1

The first dataset involves more than five hundred articles from Wikipedia and more than a hundred thousand question-answer pairs. In this dataset there is an answer for every question in the paragraph and it is relatively simple for the existing models to reach very good results with this dataset version. This idea was good in order to have a baseline, but at Stanford university it was quickly understood that it would have been better to create a more challenging version. So, less than 2 years later, the second version of SQuAD came out.

4.2.2 SQuAD v2

SQuAD v2 is an extension of the previous version which contains also non-answerable questions. Fifty thousand questions posed to be answerable-like were added by crowdworkers. This means that the dataset is made of 2/3 answerable questions and 1/3 ones. To achieve good results with this dataset, the system should also determine which question cannot find its answer in the text. This version is definitely more complicated than the previous one because the system has to analyze the whole paragraph to determine if it is possible to find an answer for a question. For this reason, the models which perform well on SQuAD v1.1 perform significantly worse on this version. Both versions have the same structure, except for the fact that in the second version two keys for each question are added: **is_impossible** and **plausible_answers**. The last one appears only in case of true value on the *is_impossible* (an example of both structure is given in A.1 and A.2).

4.2.3 OLP Dataset

This dataset is, as we have already seen, the result of a workgroup at Bielefeld University. This dataset is composed of a collection of post-game comments from football matches. The difference between the SQuAD dataset and the OLP dataset, except for the structure, is that the first one has a so called open-domain, while the OLP dataset uses a closed-domain instead.

The characteristic of this dataset is that it was thought to consider not only questions with a specific answer in the text, but also situations in which more complex techniques of text comprehension are needed. For example, we can find questions like *"How many goals did the player X score?"*, so that the answer could be structured in this way; *"Player X scored at 33'", "Player X scored again for the guest team"* and so on; the model should use some ontology to understand the relation between the answers and the questions.

Preprocessing and Conversion

Both datasets had completely different structures. For this reason, the fastest way to work with our models and the OLP dataset was to convert it in a SQuAD-like version. A python script was created to extract the text from the csv file. A file for each article by using the information about the number of characters and the sentence number (e.g. B.1), together with the annotation and question (e.g. in B.3 and in B.2) files (they were equally named, so we could easily deal with the association between them) were used to obtain the same structure as in SQuAD at the end.

Since the dataset is still under development at the University, it is not available to the public and we have not released the code yet. With the cooperation of Matteo Del Vecchio and the support of Frank Grimm, we started to study the datasets deeply. Although the datasets were actually completely different in structure, we were able to figure out what the common traits between OLP and SQuAD are.

During this analysis we discovered additional problems and part of the dataset had to be modified. In particular, we worked on problems related to the handmade annotations and fixed some policy which impeded the conversion to SQuAD.

In its first version we found different duplications in the annotations files, so we focused our attention on removing all of them and removed redundancies in the dataset by using a python script to automatize the process as much as possible. For unexpected cases we had to interfere manually. The second problem derived from the reason why OLP Dataset was born (to have a Question Answering Dataset where the answer had to be the result of some semantic inference, so the answer could not appear directly in the text and could also be composed of different pieces of text collocated far from each other in the article). We often had to deal with this problem, for example we found answer posed in the form “1:0”, to refer to a match result, while in the text the answer was “1-0”, impeding our model to learn by evaluating the answer since they were different. The question file was unique for all of the datasets and, through an id system, we could understand what question an answer referred to. But some of these questions were posed in a general way by omitting a part of the text, for example a name, as we have already shown above. Firstly, we tried to just ignore them but we quickly realized that the dataset dimension became really small. We asked our referent, Frank Grimm, how to deal with this problem and he provided us with a new version with a question file for each article in which all the general questions where specified.

Although there was no information about the answerability of a question, which is really important for SQuAD, we tried to extract it from the answer file, but there was no assurance about whether the question was answerable or not. For this reason, we suggested to add a tag **answerable** in the dataset so we could simply set the tag **is_impossible**. In this way, we had just to invert the True or False value in **answerable**.

The tokenization was not always coherent because spaces were counted directly on the difference between start and end position of tokens, but sometimes spaces were treated as token. We had to normalize every situation and find a pattern to automatize this action without losing meaning in the text.

Once we solved this aspects, we could convert the dataset and obtain our SQuAD-like version of OLP. The resulting Dataset was split in two parts, the training part and the testing part, to maintain the coherence with SQuAD, but we used also the whole dataset for an evaluation on how good the results were with a model fine-tuned on SQuAD.

4.3 Metrics

The **Exact Match (EM)** and the **F1 score** are often used to measure the accuracy in the Question Answering task. The first is basically how many times the output of the model is equal to the ground truth, while the second one considers how "near" the model is when compared to the groud truth. The F1 score is an armonic mean between **precision** and **recall**. Let us introduce some definitions to better understand the F1 formula:

- TP: True Positive is when the detection is correct.
- FP: False Positive is when the model says that the prediction is correct but it is not.
- FN: False Negative is when the model gives us a negative prediction instead of the expected positive one.
- TN: True Negative represents all the predictions which are true but in which we are not interested for the current prediction.

Here we will show the recall and precision formulas based on the notions above:

$$precision = \frac{TP}{TP + FP} = \frac{TP}{AllDetection} \tag{4.1}$$

$$recall = \frac{TP}{TP + FN} = \frac{TP}{AllGroundTruth} \tag{4.2}$$

Finally, we can write the F1 formula:

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{4.3}$$

4.4 Before BERT

Before Transformers were invented there was some approach to the Question Answering problem. Here we give an example of some older models, precursors of the more recent BERT.

4.4.1 BiDAF

BiDAF (Bi-Directional Attention Flow) [34] is a closed-domain QA model which can only answer questions with a string in the text. It was the State of Art before ELMO and BERT. This model is composed of 3 main parts: the Embedding layers, the Attention and modeling layers and the output layer.

The Embedding layers are three different levels of embeddings: character, word and phrases embedding. As we have already said, this part transforms words information into their real-valued vectors representation.

The Attention and model layers add context about the query by using additional information and merge everything in a unique output, which is called “Query-aware Context representation” in the paper.

The last layer transforms all the information received from previous layers into probabilities values, which will be used to calculate the start and end positions of an answer.

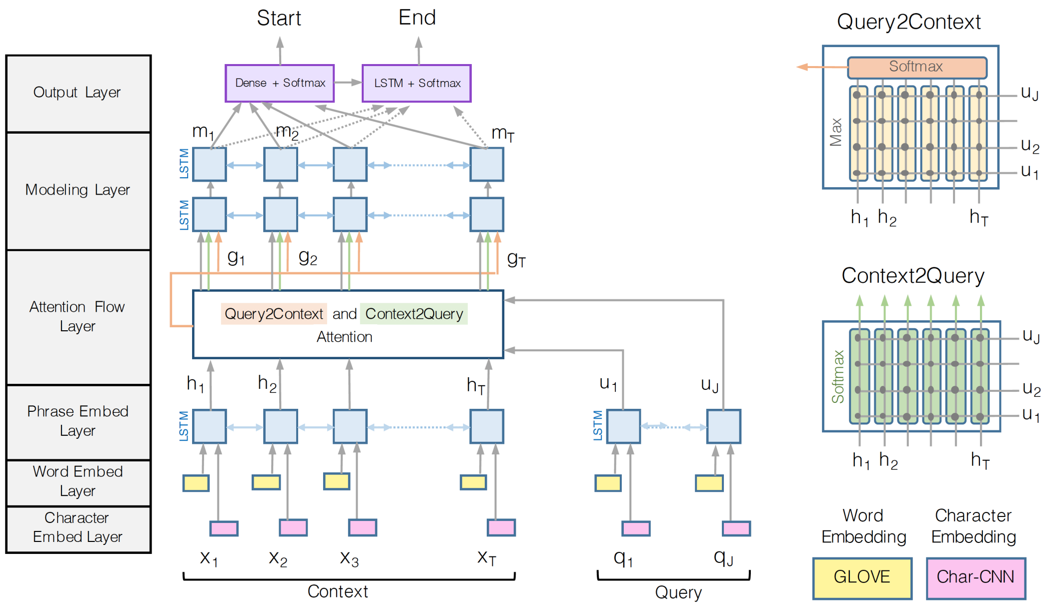


Figure 4.2: BiDAF

4.4.2 ELMo

Not only does the idea of ELMo (Embeddings from Language Models) [28] take into consideration the following word that can appear in a sentence given the current word, but also the previous one, obtaining different embeddings for the same word. It is the first time this kind of contextual approach has been used, while before a fixed embedding was assigned to each word. To do that ELMo needs to be trained on a huge dataset and it uses a bi-direction LSTM (sec **??**). The first direction is forward and considers information about the following word, while the second network goes backward to consider the previous word.

ELMo was used for Question Answering on SQuAD v1.1 by adding it to an improved version of BiDAF. This led to an improvement of 4.7 percents compared to the baseline, as shown in the relative paper.

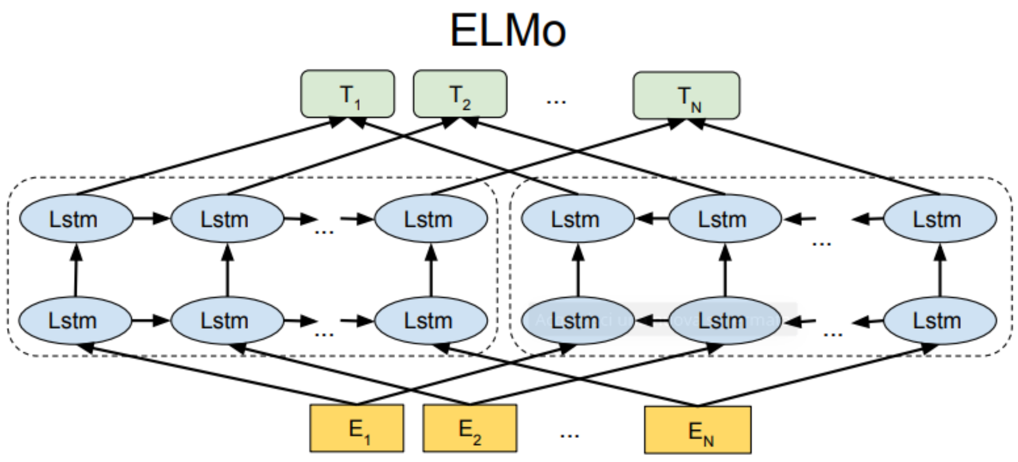


Figure 4.3: ELMo

4.5 Results Comparison

In the next table 4.1 a little recap. We will give an overview of all the most significant results published in the papers of each model:

Model	*SQuADv*1.1	*SQuADv*2
BiDAF	68.0/77.3	−/−
ELMo	−/85.8	−/−
$BERT_{LARGE}(single)$	84.1/90.9	78.7/81.9
RoBERTa	88.9/94.6	86.5/89.4
$ALBERT_{XXLARGE}$	88.3/94.1	85.1/88.1

Table 4.1: The most significant results in Question Answering, given with EM/F1 metrics

Chapter 5

DistilBERT

5.1 Models Compression

From the literature we learn that there are many techniques to reduce a model dimension and compress it to a smaller form losing something in precision but gaining in lightness and speed. We are more interested in Knowledge Distillation since it is the technique used to train DistilBERT5.3. We are going to explain it more in details in the next section 5.2.

5.1.1 Pruning

This is a technique to prune weights that match a certain criterion by assigning the value zero to them. The most common pruning criterion consists in comparing an absolute value to a threshold. If that value is smaller than this threshold, it is set to zero, because a low value contribute only a little for the final results and can be directly removed. This method was born to contrast the over-parametrizetion of models and redundancy in logic and features. There are two ways to do pruning: *one-shot pruning* and *iterative pruning*. The first one is done only once, while the second one of course more than once, but changing the pruning criterion every time (iteration)[1].

[1]`https://nervanasystems.github.io/distiller/pruning.html#pruning`

5.1.2 Quantization

Quantization reduces the precision of the values from the most widespread FP32 (32-bit floating point) to a half precision FP16, or even to a lower representation like 8-bit integer, but preserving the same accuracy. This also leads to a reduction in terms of storage, estimated to be 8 times smaller by using 8-bit integer instead of 32-bit floating point[2].

5.1.3 Regularization

"Any modification we make to a learning algorithm that is intended to reduce its generalization error, but not its training error."[13]. We are not really interested in this method, so we refer to the link at the bottom of the page[3] for a deeper study.

5.2 Knowledge Distillation

Knowledge Distillation (KB) could be seen as a transfer learning technique, even though it has a different aim. In this case we want to obtain a smaller network from the pre-trained one by transferring not its exact weights but its way to generalize the task. It is more correct to say that Knowledge Distillation's goal is something more than barely transferring knowledge, but it is also a form of compression from a huge high precision model to a smaller one, without losing too much in generalization.

It is a *reinforcement learning* technique which wants to reduce the dimension of huge models (like BERT) and the training time by transferring knowledge between two models. In particular, **Teacher-Student** paradigm is composed of a small network, the **Student**, and a bigger one, the **Teacher**, which should be trained on the complete dataset.

The training phase usually needs a consideably amount of time for models

[2]`https://nervanasystems.github.io/distiller/quantization.html`
[3]`https://nervanasystems.github.io/distiller/regularization.html`

like BERT and also its dimension and response time do not help to execute and use them in edge devices. The final aim is to teach the **Student** how to simulate the **Teacher's** behaviour. Let us define the two behaviour as φ^T and φ^S .

In the case of the transformer distillation MHA (Multi-Head Attention) and the FFN (Feed-Forward Network), the output could be used as behaviour functions. The Knowledge Distillation can be modeled by minimizing the following function:

$$L_{KD} = \sum_{x \in \chi} L(\varphi^s(x), \varphi^t(x)) \tag{5.1}$$

where χ is the dataset, x the next text input and *L(.)* is the loss function destinated to evaluate the differences between the student and the teacher predictions. We will introduce, explaining more in details the distillation, notions like **Student-Teacher network** 5.2.1, softmax with **Temperature** 5.2.2 and **Dark Knowledge** 5.2.3.

5.2.1 Teacher-Student

As it was already anticipated the **Teacher** is the bigger high precision network used to transmit the "behaviour" to the **Student** network. Firstly, the Teacher network has to be trained over the complete dataset with high performances. Secondly, when a Student is built, there has to be correspondence between intermediate levels since they are different in dimension. The pipeline is quite simple and it can be summarized in this way: the same input passes through both models, the Teacher and the Student, producing their output that is fed to the softmax. This is a key point because here we find a special version of softmax, the softmax-temperature (5.2.2), which produces soft-labels from the output of the Teacher and soft-predictions from the Students output. Then the Loss function is calculated between them. The same Temperature is used and this part is the **Distillation Loss**. Also the hard prediction of the Student model is calculated by using the normal softmax, which will be compared to the **Ground Truth** obtaining the Student Loss.

Finally, both results are summed together to obtain the final Loss:

$$L(x;W) = \alpha * H(y, \sigma(z_s;T=1)) + \beta * H(\sigma(z_t;T=\tau), \sigma(z_s, T=\tau)) \quad (5.2)$$

Where x is the input of the model, W are the student's weights, σ is the softmax (also with the **Temperature** T), y are the labels of the **Ground Truth** and H is the Cross-Entropy loss, while z represents both Student and Teacher logits. The following picture (5.1) shows graphically the formula 5.2 above:

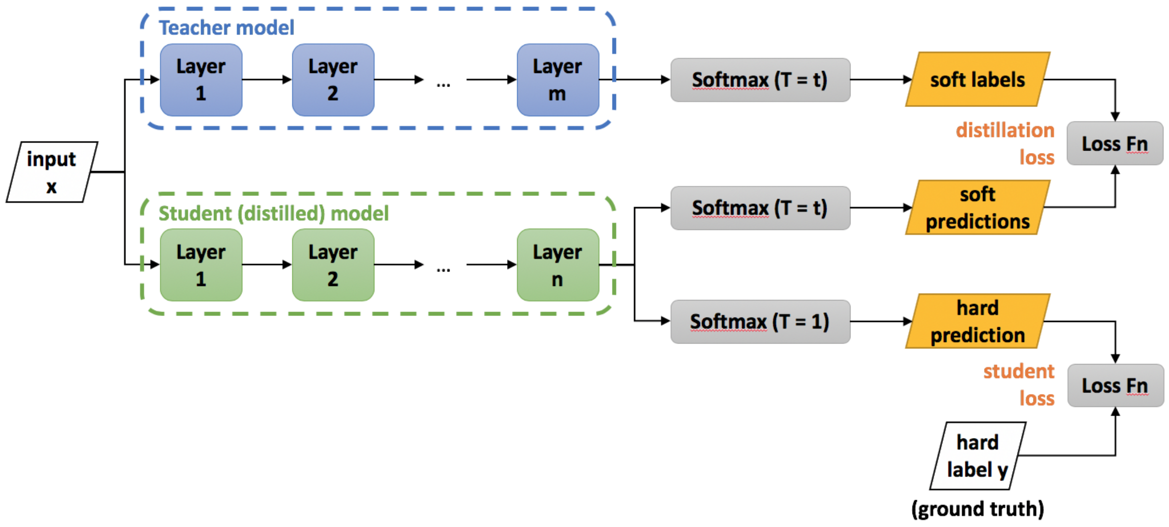


Figure 5.1: Teacher-Student Model schema [20]

5.2.2 Softmax-Temperature

Softmax-temperature is a standard Softmax with a coefficient that makes the predictions of a network "softer". In fact, if we compare the two formulas we can easily notice that the only difference is in this T, which divides the network logits.

$$p_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \quad (5.3)$$

$$p_i = \frac{exp(\frac{z_i}{T})}{\sum_j exp(\frac{z_j}{T})} \quad (5.4)$$

Where the 5.3 is the standard one and the 5.4 is the Softmax with temperature.
This is the baseline of the *Dark Knowledge* in explained in the next section (5.2.3).

5.2.3 Dark Knowledge

Dark Knowledge is the result obtained from the application of Softmax-Temperature to the Teacher network prediction, adding more information to the class that the Teacher found to be more probable. This additional knowledge is the so-called **Dark Knowledge** [21] that we want to transfer from the Teacher network to the Student. As we have already said in the previous section 5.2.1, the soft-prediction of the Student is done by using the same value of Temperature.

5.3 DistilBERT Model

Since BERT is a mastodontic model and requires a lot of time to train and a significantly computational effort, the HuggingFace research group [12] model tried to combine the Knwoledge Distillation with BERT. In addition, the token-type embeddings and the pooler were removed, because they realized that the next sentence classification was not so effective. The rest of the architecture was kept identical but the number of layers was reduced of a factor of two. As a result, a smaller language model, DistilBERT, was born. This was trained with BERT supervision compressing it and preserving almost the same performance as BERT.

To transfer the knowledge in DistilBERT, this was trained on the soft target probability of the Teacher:

$$L_{CE} = \sum_i t_i * log(s_i) \tag{5.5}$$

t_i and s_i are calculated by using a *softmax-temperature* 5.4, where T is always *Temperature* which controls the smoothness of the output distribution and was set to the same value, both for Teacher and the Student at training time, while at inference time it is set to 1 (the standard softmax), to calculate the hard prediction of the student. The code below shows how the knowledge was distilled during the fine tuning phase of DistilBERT. The formula is calculated in the *run_squad_w_distillation.py*[4]:

Listing 5.1: Loss with distillation used during Fine-Tuning

```
loss_fct = nn.KLDivLoss(reduction="batchmean")
loss_start = loss_fct(
    F.log_softmax(start_logits_stu / args.temperature,
       dim=-1),
    F.softmax(start_logits_tea / args.temperature, dim
       =-1),
) * (args.temperature ** 2)
loss_end = loss_fct(
    F.log_softmax(end_logits_stu / args.temperature, dim
       =-1),
    F.softmax(end_logits_tea / args.temperature, dim=-1)
       ,
) * (args.temperature ** 2)
loss_ce = (loss_start + loss_end) / 2.0
loss = args.alpha_ce * loss_ce + args.alpha_squad * loss
```

while the following code comes from the file *distiller.py* in the same repository:

Listing 5.2: Loss with distillation used during Training

```
loss_ce = (
    self.ce_loss_fct(
        F.log_softmax(s_logits_slct / self.temperature,
           dim=-1),
```

[4]https://github.com/huggingface/transformers.git

```
        F.softmax(t_logits_slct / self.temperature, dim
           =-1),
    )* (self.temperature) ** 2
)
loss = self.alpha_ce * loss_ce
```

According to the results published on the DistilBERT paper [33], it can reach 97% of BERT understanding skills, despite being 40% smaller and 60% faster. For completeness, these results are summed up in the table 5.1 with the GLUE baseline [33].

$Model$	$Score$	$CoLA$	$MNLI$	$MRCP$	$QNLI$	QQP	RTE	$SST-2$	$STS-B$	$WNLI$
$ELMo$	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
$BERT-base$	77.6	48.9	84.3	88.6	89.3	89.5	71.3	91.7	91.2	43.7
$DistilBERT$	76.8	49.1	81.8	90.2	90.2	89.2	62.9	92.7	90.7	44.4

Table 5.1: Comparison of DistilBERT's results with BERT-base's and ELMo performance

DistilBERT can preserve most of the results of BERT-base on SQuAD v1.1 or about 2 point less and even better, adding an ulterior step of distillation during the fine-tuning phase. The results are shown in the next table (5.2), where the double distillation process is indicated by a (D). Both models are incomparable in terms of lightness, since DistilBERT has 66 million of parameters against the 110 of $BERT_{BASE}$. It gets good results also on inference time because BERT-base needs about 668 seconds, while DistilBERT 410 seconds 5.2.

$Model$	$IMDb$	$SQuAD(EM/F1)$
$BERT-base$	93.46	81.2/88.5
$DistilBERT$	92.82	77.7/85.8
$DistilBERT(D)$	–	79.1/86.9

Table 5.2: DistilBERT ability of preserving the results of BERT

$Model$	$Parameters(millions)$	$Inference(seconds)$
$ELMo$	180	895
$BERT - base$	110	668
$DistilBERT$	66	410

Table 5.3: DistilBERT number of parameters and speed

One further advantage of having less dimension and inference time is that we can execute the model on edge devices (that could be a smartphone), which do not have enough memory and computational power to execute the whole BERT in an acceptable amount of time. The HuggingFace team also managed to build an android application with the tensorflow version of DistilBERT for question answering[5].

[5]https://github.com/huggingface/tflite-android-transformers.git

Chapter 6

QADistilBERT

6.1 QABERT

This version of Question Answering level was born from the collaboration with Matteo Del Vecchio during the "projekt" at Bielefeld University, which had the aim to start getting our hands on the existing NLP models.

QABERT is a task specific implementation of question answering level for BERT. We thought about what could be the benefit of developing a different architecture of this layer. For this reason, we implemented 5 combinations of fully connected layers and activation functions, in total, three models with 2 fully connected layers, each of them with an activation function among ReLU, GELU and Tanh, and two models with 4 fully connected layers with GELU or ReLU between couple of layers. In the following sections we will show come images we have created to have a graphical look at their compositions.

6.1.1 QABERT Vanilla

This was only to have a BERT-like baseline, since we did not have access to the same hardware of google. We fine-tuned the vanilla version just like the BERT QA level was developed by google. It does not use any activation function and has an input shape (*batch_size, max_seq_length, hidden_size*) and output (*batch_size, max_seq_length, 2*).

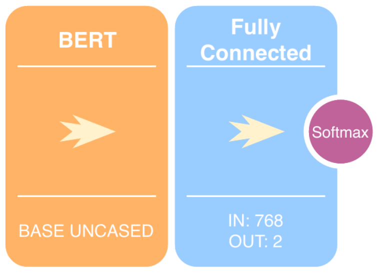


Figure 6.1: QABERTVanilla model schema

6.1.2 QABERT 2L

To explore the benefit of having more layers, we added two layers on the top of BERT and tried different activation functions to understand which one works better. We did this in order to achieve better results on question answering. We also tried different configuration shapes and we will present only the best ones. The idea was to have a conic function reduction of the dimension; in this way we could have a more gradual reduction from the BERT output dimension to 2.

QABERT 2L Tanh

In the version with **Tanh** function we opted for a reduction, firstly, from 768 to 384, and then from 384 to 2 (Figure 6.2).

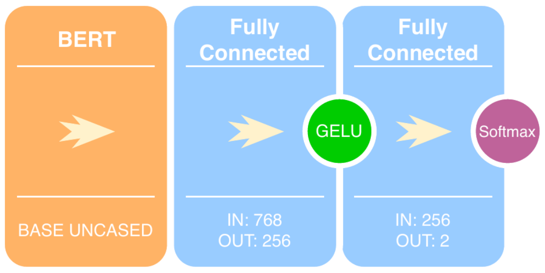


Figure 6.2: QABERT2LTanh model schema

QABERT 2L GELU

The second version of this model uses the GELU (Gaussian Error Linear Units) activation function, which is used in BERT [14]. The conic in this case decreases from 768 to 256 and from 256 to 2 (Figure 6.3).

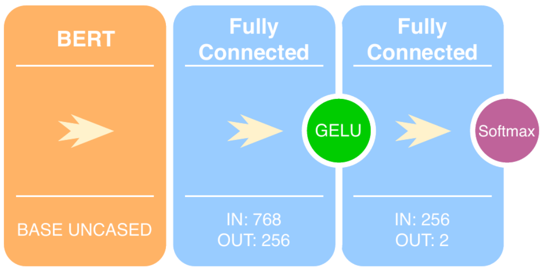


Figure 6.3: QABERT2LGELU model schema

QABERT 2L ReLU

With ReLU activation function the same input dimension was kept in both layers since the skip idea was introduced. The output of BERT is fed to the first layer and then added to its output before feeding everityng to the second layer (Figure 6.4).

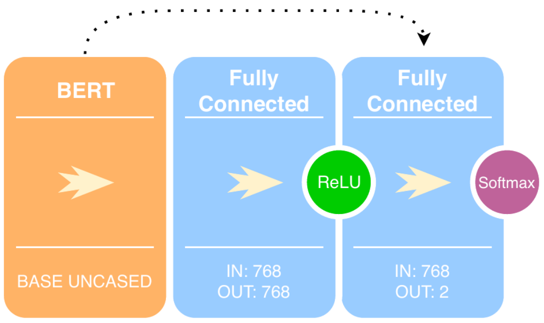


Figure 6.4: QABERT2LReLU model schema

6.1.3 QABERT 4L

With four layers we were able to develop a different idea of shapes reduction. In this case we had more layers and we thought about a more "rhombic" idea. In fact, the shape increases its dimensions between the first two layers and uses the same conic concept as above for the last layers.

QABERT 4 ReLU

This model uses the ReLU activation function between the layers and the shape, as we have already said, which are firstly enlarged from 768 to 1024 and reduced from 1024 to 200 and from 200 to 2 at the end (Figure 6.5).

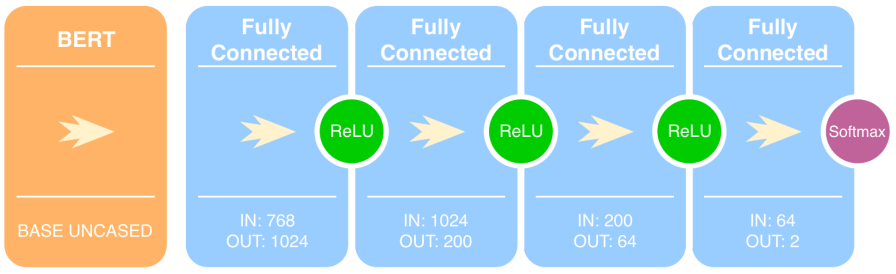


Figure 6.5: QABERT4LReLU model schema

QABERT 4 GELU

We will introduce another idea which consists in using to use only the "rhombic" dimensions changing. The skip idea wants to take more into account the BERT output. The third layer input thus has a dimension equal to BERT output. As it is shown in the next picture, the original BERT output is added to the second fully connected layer's output. We will use this question answering level as a baseline for our study because, looking at the results we obtained by fine-tuning BERT with different configurations, this structure reached better results with respect to the others (Figure 6.6).

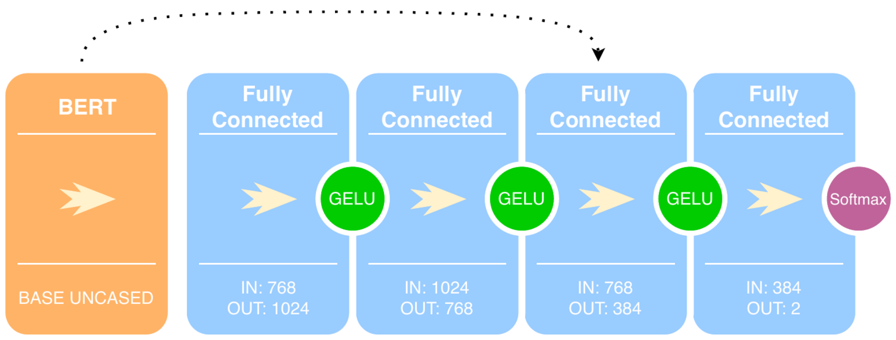


Figure 6.6: QABERT4LGELUSkip model schema

6.2 Implementation

It was used as starting code the HuggingFace interface of Transformers[1] as baseline for our work, in particular it was necessary to modify the functions ***DistilBertForQuestionAnswering*** and ***BertForQuestionAnswering*** in the same way. Considering the substantial changes, we decided to rewrite the functions and rename them ***QABERT4LGELUSkip*** and ***QADistilBert4LGELUSkip*** and put them in a separate file independent from the transformers repository, but doing it required to execute our job. For the training and evaluation task we used a computer equipped with 2 GPUs Nvidia P100[2] with 16 GB of memory and another one equipped with 2 GPUs Nvidia 1080Ti[3]. Both computers are part of a cluster, which is collocated in the CITEC area of Bielefeld University and composed of 6 nodes for a total of 4 Nvidia P100 and 8 Nvidia 1080Ti.

The next piece of code comes from the file *modeling_distibert.py* and shows the implementation of a single question answering layer:

Listing 6.1: Original BERT QA level with one layer (from BERT code)

```
self.bert = BertModel(config)
```

[1]https://github.com/huggingface/transformers.git
[2]https://www.nvidia.com/en-us/data-center/tesla-p100/
[3]https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti/

```
self.qa_outputs = nn.Linear(config.hidden_size, config.
    num_labels)
```

Listing 6.2: Output generation until logits (from BERT code)

```
sequence_output = outputs[0]
outputs = self.bert(input_ids, attention_mask=
    attention_mask, token_type_ids=token_type_ids,
    position_ids=position_ids, head_mask=head_mask,
    inputs_embeds=inputs_embeds)
logits = self.qa_outputs(sequence_output)
start_logits, end_logits = logits.split(1, dim=-1)
start_logits = start_logits.squeeze(-1)
end_logits = end_logits.squeeze(-1)
outputs = (start_logits, end_logits,) + outputs[2:]
```

While following the implementation of ***QADistilBERT4LGELUSkip*** (which is almost the same for ***QABERT4LGELUSkip***), the model output passes through 3 intermediate levels before output logits, which will be of dimension $B_S\ x\ EMB_S\ x\ 2$. It is subsequently split in two different vectors of dimension $B_S\ x\ EMB_S\ x\ 1$. Then a squeeze operation reduces the dimension in $B_S\ x\ EMB_S$

Listing 6.3: QADistilBert4LGELUSkip level with one layer

```
self.distilbert = DistilBertModel(config)
self.middleOut1 = nn.Linear(config.dim, 1024)
self.middleOut2 = nn.Linear(1024, 768)
self.middleOut3 = nn.Linear(768, 384)
self.qa_outputs = nn.Linear(384, config.num_labels)
self.dropout = nn.Dropout(config.qa_dropout)
```

Listing 6.4: QADistilBERT4LGELUSkip level with one layer

```
distilbert_output = self.distilbert(input_ids=input_ids,
    attention_mask=attention_mask, head_mask=head_mask,
   inputs_embeds=inputs_embeds)
hidden_states = distilbert_output[0]  # (bs,
   max_query_len, dim)
midOut1 = self.dropout(gelu(self.middleOut1(
   hidden_states)))
midOut2 = self.dropout(gelu(self.middleOut2(midOut1)))
midOut3 = self.dropout(gelu(self.middleOut3(midOut2 +
   hidden_states)))
logits = self.qa_outputs(midOut3)  # (bs, max_query_len,
    2)
start_logits, end_logits = logits.split(1, dim=-1)
start_logits = start_logits.squeeze(-1)  # (bs,
   max_query_len)
end_logits = end_logits.squeeze(-1)  # (bs,
   max_query_len)
outputs = (start_logits, end_logits,) +
   distilbert_output[1:]
```

The vanilla QA level is almost the same in DistilBERT implementation, so we decided to take it as an example code from the two models to emphasize the fact that the pooler was removed in DistilBERT. We can notice, in the last line of both code 6.2 and 6.4, that **outputs** start from 2 instead of 1 as it in DistilBERT. Since the pooler output is useless in Question Answering task, in BERT implementation it has to be removed.

6.3 Results

The aim of this work was to understand if some benefit could be obtained by adding fully-connected layers to the question answering level. Looking at the pictures in the first section of this chapter, it is easily understandable how this level was structured. In other words, BERT has been treated as a black box, so the focus is on the specific task of question answering.

Firstly, a fine-tuning over $BERT_{BASE}$ was performed by using SQuAD v2 to have a baseline, since there are no official results on this task for the BASE version of BERT. Then we created the QABERT4LGELUSkip and fine-tuned with the same hype-parameters as well, outperforming the baseline results. We measured the F1 score 4.3 and the EM (Exact Match) score to have the same set of results as in SQuAD paper. According to the paper, the F1 score measures the average overlap between the prediction and the ground truth answer, while EM measures the percentage of predictions that match exactly everyone of the ground truth answers [30].

The configuration highlights are on the batch size. This was set on 12 and a gradient accumulation at 24. We fine-tuned both for 3 epochs without performing any warmup period and with a learning rate of 5E-5.

QABERT4LGELUSkip		BERT-QA-Vanilla	
$F1$	EM	$F1$	EM
76.56	73.15	69.38	66.074

Table 6.1: Comparison between fine-tuning on $BERT_{BASE}$ with a 4 Layer QA level and 1 Layer QA level using SQuAD v2 Dataset

Unfortunately, due to hardware constraint, it was impossible to do any try on $BERT_{LARGE}$ fine-tuning. In addition, it was impossible to distill completely a new smaller version of BERT from scratch, to use a combination of datasets and to use the theory of RoBERTa, which says that BERT is still undertrained. In the end, we focused our effort on a smaller improvement in question answering only.

6.4 SQuAD Results

Once the baseline was developed, we thought about making the same improvement in DistilBERT and fine-tuning it by using QABERT4LGELUSkip as the Teacher for the distillation part. Moreover, the same level for the smaller model was created, called QADistilBERT4LGELUSkip, which is the same idea of the Teacher; this was also fine-tuned. The final objective was to compare the results of both versions on this specific task (QA). The next table collects all the six fine-tuning-distillations we did, the highlighted line is the one that allowed us to get better results:

Configurations	Learning Rate	Batch Size	Num Epochs	Warmup	Grad Acc steps
$config1$	$3.00E-05$	12.00	3	10% total steps	0
config2	**5.00E-05**	**12.00**	**4**	**10% total steps**	**0**
$config3$	$3.00E-05$	24.00	3	10% total steps	0
$config4$	$5.00E-05$	32.00	3	10% total steps	0
$config5$	$5.00E-05$	32.00	6	10% total steps	0
$config6$	$5.00E-05$	32.00	4	10% total steps	0

Table 6.2: Hyperparameters configurations for fine-tuning distillation step

We did not expect significant improvements because of our limited resource. Despite this, we got some interesting results. Starting from the **Teacher** we obtained an unexpected increase in about 7 points, both in F1 and Exact Match scores, as it shown in the table 6.1. We believe that this is already a good result, but it is still useless if we are looking for a model which has to be executed on edge devices. For this reason, and being inspired by DistilBERT work, we thought we could get better results by using our QABERT4LGELUSkip as teacher, firstly by fine-tuning the Vanilla DistilBERT QA layer and then by using the 4 layers implementation.

We summarized the results obtained from all trainings in the following table in order to have a general overview for all the configurations 6.3: Differently from the paper, we chose to use the version 2 of SQuAD dataset and while normally the higher the scores are, the higher the value of *batch_size* is, in our case the best results came from the configuration with *batch_size* of 12. Other parameters which demonstrated to have an impact during the training are the *learning_rate* and the *number_of_epochs* (this is a very difficult choice because it could lead to overfitting if the model is trained for too long, and

Models	*DistiBERT*		*QADistilBERT*	
Configurations	F1	EM	F1	EM
config1 (lr=3.00E-05, bs=12, epochs=3)	69.96	66.45	70.71	67.41
config2 (lr=5.00E-05, bs=12, epochs=4)	**71.07**	**67.69**	**71.84**	**68.58**
config3 (lr=3.00E-05, bs=24, epochs=3)	68.62	65.04	69.71	66.31
config4 (lr=5.00E-05, bs=32, epochs=3)	70.04	66.68	71.28	67.94
config5 (lr=5.00E-05, bs=32, epochs=6)	69.99	66.61	70.6	67.24
config6 (lr=5.00E-05, bs=32, epochs=4)	70.77	67.55	70.22	66.72

Table 6.3: QADistilBERT results after 6 fine-tuning configurations

to be underfitting otherwise).

In conclusion, during our experiments we found out that a wrong value of *gradient_accumulation_steps* leads to bad results, so we thought it is better to set it to zero, which means not using it.

6.5 OLP Results

We investigated also the results with the OLP dataset through a set of experiments. We verified the behavior of our models with and without fine-tuning on OLP. Firstly, we used our best configuration of DistilBERT and QADistilBERT4LGELUSkip without fine-tuning on OLP (in two versions, one of 384 and one of 512 of maximum sequence length) and by using the whole dataset to make predictions on it. Results that are summarized in the next table 6.4: In our second experiment we split the OLP Dataset in two,

	Dim	F1	EM	Ans F1	Ans EM	No Ans F1	No Ans EM
DistilBERT	384	50.13	49.28	12.75	10.51	72.53	72.53
	512	49.93	49.12	12.77	10.61	71.89	71.89
QADistilBERT4L	384	51.04	50.60	8.03	6.86	76.81	76.81
GELUSkip	512	50.85	50.31	6.06	7.52	76.45	76.45

Table 6.4: The results obtained from a simple evaluation on the whole OLP Dataset

train and dev, and we made it for both versions, the 384 and 512 maximum sequence length. Then we just used the train part for fine-tuning, for 4 epochs more, the same versions of our model as above. It is interesting to notice

	Dim	F1	EM	Ans F1	Ans EM	No Ans F1	No Ans EM
DistilBERT	384	65.97	64.96	20.59	17.65	89.47	89.47
	512	66.94	66.04	21.30	28.68	90.60	90.60
QADistilBERT4L	384	65.01	63.50	18.32	13.90	89.19	89.19
GELUSkip	512	65.22	63.60	19.03	14.28	89.17	89.17

Table 6.5: The results obtained from a simple evaluation obtained after a 4 epochs fine tuning on OLP

that there is a substantial increment in general F1 and Exact Match scores after the fine-tuning on OLP. It is also interesting that the predictions which contribute more to the general results are the ones about non-answerable questions. We actually expected this kind of behaviour due to the stucture of the dataset. How the questions were posed was not thought to solve the task of finding a span which contains the answer directly in the text, so there is an exiguous number of questions which can be found in this way.

Conclusion

This work turned out to be really interesting because, thanks to the HuggingFace interface for BERT, it was quite simple to add our changes. This was a good opportunity to put hands on a model like BERT and not just use it as a black box, even though it was only for the specific task of question answering. As it has been already said, the first good result is an increase of 7 point in F1/EM score from the baseline.

Looking at the results in the table 6.3 we can observe that the student that uses a final level with 4 fully-connected layers and GELU activation function in between each of them, in general, reaches better results than the one that uses Vanilla implementation (only one fully-connected layer without activation functions). Using SQuAD v2 we can notice that with respect to the F1/EM scores of
QABERT4LGELUSkip, QADistilBERT4LGELUSkip preserves the 94% in F1 and 93,75% in EM. The only case in which we got better results with the Vanilla QA of DistilBERT was the *config6*, where the **batch_size** is 32 and was trained for 4 epochs.

In conclusion (as it could be seen) despite the limited hardware, we got a general improvement by using this extension of the question answering level; this work gives space to further expansion and other optimization, hoping to access to a more performant hardware. All the experiments can be reproduced by downloading the github repository QADistilBERT4LGELUSkip[4]. Our policy consists in sharing, so the entire code is an open source and is

[4]`https://github.com/simonepreite/QADistilBERT4LGELUSkip`

provided under MIT Licence.

6.6 Future Works

The same strategies could be applied with a bigger version of BERT, like $BERT_{LARGE}$. An example is to have the opportunity to train, by transferring knowledge with distillation to an even smaller model than DistilBERT, and to study the precision it could reach by taking less inference time. If the resulting model is small enough in terms of dimension and speed, it can be executed on an edge device. However, with a view to having smaller and smaller devices but, at the same time, not having always an internet connection, models that do not require high performances could be integrated directly in apps, losing a little bit in precision, if an hosted bigger model could not be reached. As additional idea, would be a little bot that could use BERT to build a bot on diverse platforms, like Telegram or Facebook chats. In this case both version, small or big, can be used because the model has to be executed server-side (with an higher performance). The choice depends on whether we are looking for a quick response time or for better precision.

Appendix A

SQuAD examples

A.1 SQuAD v1.1

Listing A.1: SQuAD v1.1 example

```
{
  "title": "University_of_Notre_Dame",
  "paragraphs": [
    {
      "qas": [
        {
          "answers": [
            {
              "answer_start": 381,
              "text": "a Marian place of prayer and reflection"
            }
          ],
          "question": "What is the Grotto at Notre Dame?",
          "id": "5733be284776f41900661181"
        },
        {
          "answers": [
            {
              "answer_start": 92,
              "text": "a golden statue of the Virgin
                 Mary"
            }
          ],
          "question": "What sits on top of the Main
             Building at Notre Dame?",
          "id": "5733be284776f4190066117e"
        }
      ],
      "context": "Architecturally, the school has a
         Catholic character. Atop the Main Building's
         gold dome is a golden statue of the Virgin Mary
         . Immediately in front of the Main Building and
          facing it, is a copper statue of Christ with
         arms upraised with the legend \"Venite Ad Me
         Omnes\". Next to the Main Building is the
         Basilica of the Sacred Heart. Immediately
         behind the basilica is the Grotto, a Marian
         place of prayer and reflection. It is a replica
          of the grotto at Lourdes, France where the
         Virgin Mary reputedly appeared to Saint
         Bernadette Soubirous in 1858. At the end of the
          main drive (and in a direct line that connects
          through 3 statues and the Gold Dome), is a
         simple, modern stone statue of Mary."
    }
  ]
}
```

A.2 SQuAD v2

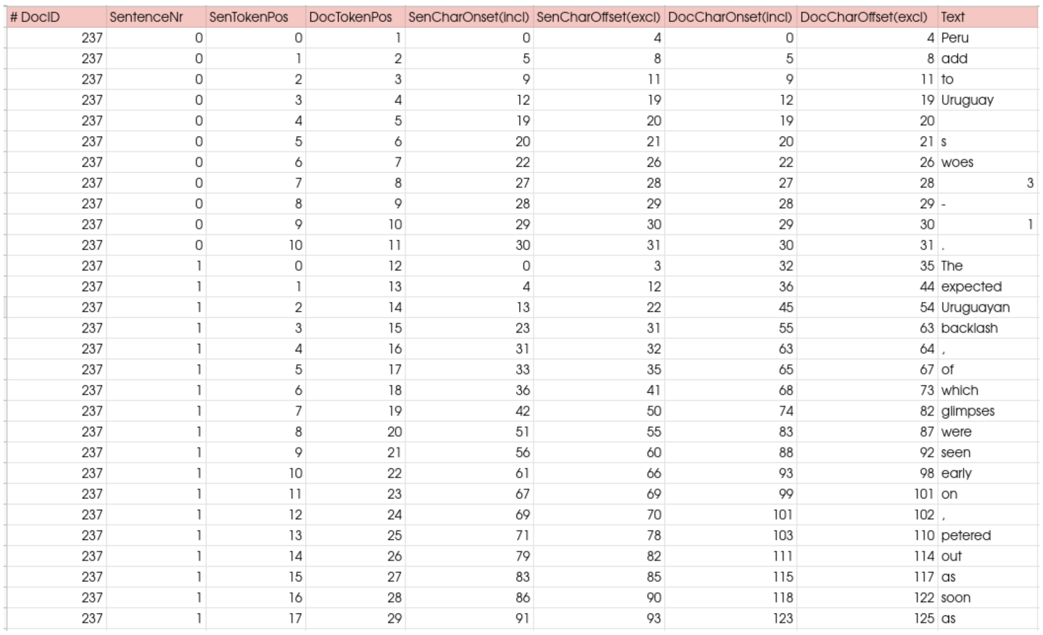
Listing A.2: SQuAD v2 example

```
{
  "title": "Separation_powers_United_States_Constitution
     ",
  "paragraphs": [
    {
      "qas": [
        {
          "question": "How many divisions of the
             government did Montesquieu call for?",
          "id": "56de244f4396321400ee25f0",
          "answers": [
            {
              "text": "three",
              "answer_start": 166
            }
          ],
          "is_impossible": false
        },
        {
          "plausible_answers": [
            {
              "text": "Separation of powers",
              "answer_start": 0
            }
          ],
          "question": "What originates in the writings
             of the Legislative powers?",
          "id": "5ad370ad604f3c001a3fe25d",
          "answers": [],
          "is_impossible": true
        }
      ],
      "context": "Separation of powers is a political
         doctrine originating in the writings of
         Montesquieu in The Spirit of the Laws where he
         urged for a constitutional government with
         three separate branches of government. Each of
         the three branches would have defined abilities
          to check the powers of the other branches.
         This idea was called separation of powers. This
          philosophy heavily influenced the writing of
         the United States Constitution, according to
         which the Legislative, Executive, and Judicial
         branches of the United States government are
         kept distinct in order to prevent abuse of
         power. This United States form of separation of
          powers is associated with a system of checks
         and balances."
    }
  ]
}
```

Appendix B

OLP Dataset

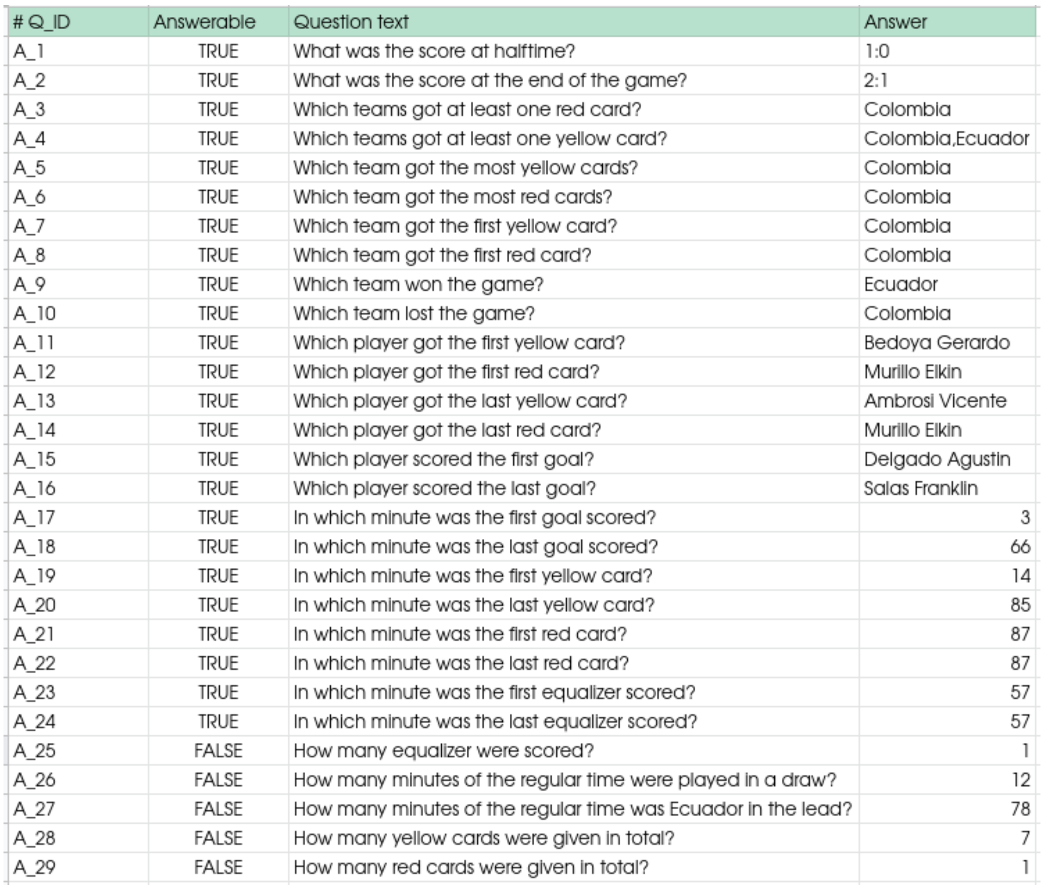
B.1 OLP Article



# DocID	SentenceNr	SenTokenPos	DocTokenPos	SenCharOnset(incl)	SenCharOffset(excl)	DocCharOnset(incl)	DocCharOffset(excl)	Text
237	0	0	1	0	4	0	4	Peru
237	0	1	2	5	8	5	8	add
237	0	2	3	9	11	9	11	to
237	0	3	4	12	19	12	19	Uruguay
237	0	4	5	19	20	19	20	
237	0	5	6	20	21	20	21	s
237	0	6	7	22	26	22	26	woes
237	0	7	8	27	28	27	28	3
237	0	8	9	28	29	28	29	-
237	0	9	10	29	30	29	30	1
237	0	10	11	30	31	30	31	.
237	1	0	12	0	3	32	35	The
237	1	1	13	4	12	36	44	expected
237	1	2	14	13	22	45	54	Uruguayan
237	1	3	15	23	31	55	63	backlash
237	1	4	16	31	32	63	64	,
237	1	5	17	33	35	65	67	of
237	1	6	18	36	41	68	73	which
237	1	7	19	42	50	74	82	glimpses
237	1	8	20	51	55	83	87	were
237	1	9	21	56	60	88	92	seen
237	1	10	22	61	66	93	98	early
237	1	11	23	67	69	99	101	on
237	1	12	24	69	70	101	102	,
237	1	13	25	71	78	103	110	petered
237	1	14	26	79	82	111	114	out
237	1	15	27	83	85	115	117	as
237	1	16	28	86	90	118	122	soon
237	1	17	29	91	93	123	125	as

Figure B.1: example of an OLP article tokenized

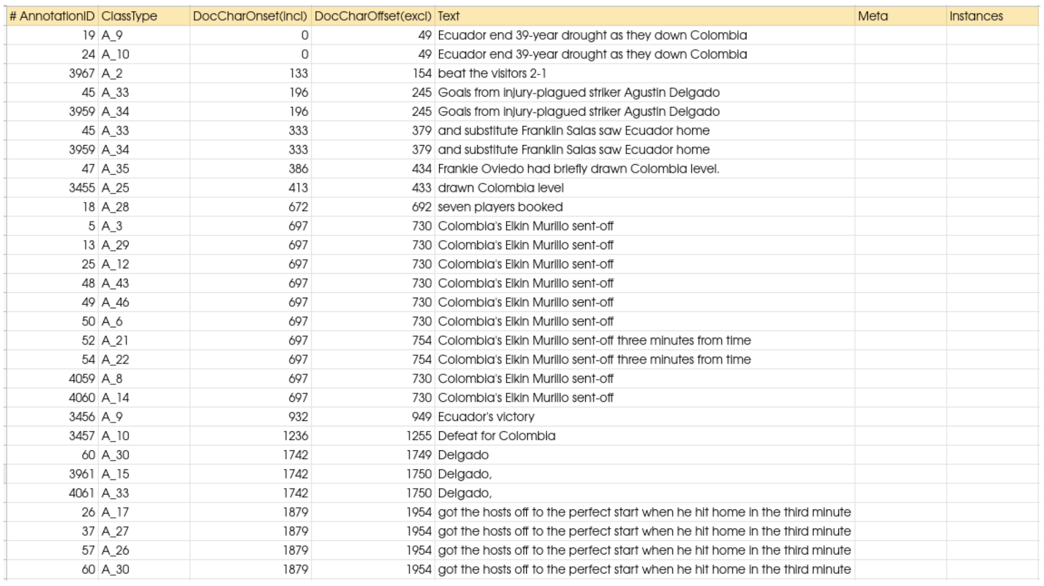
B.2 OLP Questions



# Q_ID	Answerable	Question text	Answer
A_1	TRUE	What was the score at halftime?	1:0
A_2	TRUE	What was the score at the end of the game?	2:1
A_3	TRUE	Which teams got at least one red card?	Colombia
A_4	TRUE	Which teams got at least one yellow card?	Colombia,Ecuador
A_5	TRUE	Which team got the most yellow cards?	Colombia
A_6	TRUE	Which team got the most red cards?	Colombia
A_7	TRUE	Which team got the first yellow card?	Colombia
A_8	TRUE	Which team got the first red card?	Colombia
A_9	TRUE	Which team won the game?	Ecuador
A_10	TRUE	Which team lost the game?	Colombia
A_11	TRUE	Which player got the first yellow card?	Bedoya Gerardo
A_12	TRUE	Which player got the first red card?	Murillo Elkin
A_13	TRUE	Which player got the last yellow card?	Ambrosi Vicente
A_14	TRUE	Which player got the last red card?	Murillo Elkin
A_15	TRUE	Which player scored the first goal?	Delgado Agustin
A_16	TRUE	Which player scored the last goal?	Salas Franklin
A_17	TRUE	In which minute was the first goal scored?	3
A_18	TRUE	In which minute was the last goal scored?	66
A_19	TRUE	In which minute was the first yellow card?	14
A_20	TRUE	In which minute was the last yellow card?	85
A_21	TRUE	In which minute was the first red card?	87
A_22	TRUE	In which minute was the last red card?	87
A_23	TRUE	In which minute was the first equalizer scored?	57
A_24	TRUE	In which minute was the last equalizer scored?	57
A_25	FALSE	How many equalizer were scored?	1
A_26	FALSE	How many minutes of the regular time were played in a draw?	12
A_27	FALSE	How many minutes of the regular time was Ecuador in the lead?	78
A_28	FALSE	How many yellow cards were given in total?	7
A_29	FALSE	How many red cards were given in total?	1

Figure B.2: example of questions on a OLP article

B.3 OLP Annotations



# AnnotationID	ClassType	DocCharOnset(incl)	DocCharOffset(excl)	Text	Meta	Instances
19	A_9	0	49	Ecuador end 39-year drought as they down Colombia		
24	A_10	0	49	Ecuador end 39-year drought as they down Colombia		
3967	A_2	133	154	beat the visitors 2-1		
45	A_33	196	245	Goals from injury-plagued striker Agustin Delgado		
3959	A_34	196	245	Goals from injury-plagued striker Agustin Delgado		
45	A_33	333	379	and substitute Franklin Salas saw Ecuador home		
3959	A_34	333	379	and substitute Franklin Salas saw Ecuador home		
47	A_35	386	434	Frankie Oviedo had briefly drawn Colombia level.		
3455	A_25	413	433	drawn Colombia level		
18	A_28	672	692	seven players booked		
5	A_3	697	730	Colombia's Elkin Murillo sent-off		
13	A_29	697	730	Colombia's Elkin Murillo sent-off		
25	A_12	697	730	Colombia's Elkin Murillo sent-off		
48	A_43	697	730	Colombia's Elkin Murillo sent-off		
49	A_46	697	730	Colombia's Elkin Murillo sent-off		
50	A_6	697	730	Colombia's Elkin Murillo sent-off		
52	A_21	697	754	Colombia's Elkin Murillo sent-off three minutes from time		
54	A_22	697	754	Colombia's Elkin Murillo sent-off three minutes from time		
4059	A_8	697	730	Colombia's Elkin Murillo sent-off		
4060	A_14	697	730	Colombia's Elkin Murillo sent-off		
3456	A_9	932	949	Ecuador's victory		
3457	A_10	1236	1255	Defeat for Colombia		
60	A_30	1742	1749	Delgado		
3961	A_15	1742	1750	Delgado,		
4061	A_33	1742	1750	Delgado,		
26	A_17	1879	1954	got the hosts off to the perfect start when he hit home in the third minute		
37	A_27	1879	1954	got the hosts off to the perfect start when he hit home in the third minute		
57	A_26	1879	1954	got the hosts off to the perfect start when he hit home in the third minute		
60	A_30	1879	1954	got the hosts off to the perfect start when he hit home in the third minute		

Figure B.3: example of annotations on a OLP article

Appendix C

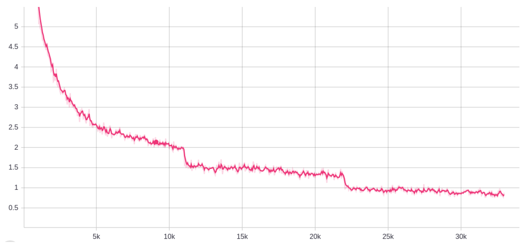
Training Summary

C.1 Configuration

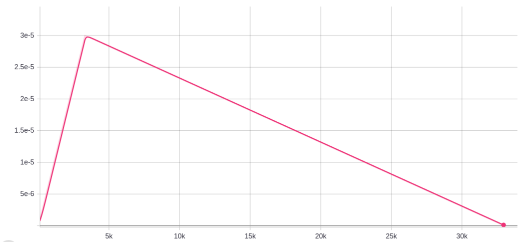
Configurations	Learning Rate	Batch Size	Num Epochs	Warmup	Grad Acc Step
$config1$	$3.00E-05$	12.00	3	10% total steps	0
config2	**5,00E-05**	**12,00**	**4**	**10% total steps**	**0**
$config3$	$3.00E-05$	24.00	3	10% total steps	0
$config4$	$5.00E-05$	32.00	3	10% total steps	0
$config5$	$5.00E-05$	32.00	6	10% total steps	0
$config6$	$5.00E-05$	32.00	4	10% total steps	0

Table C.1: Same table as 6.2, it is putted her only to have a quick configurations summary

C.2 DistilBERT

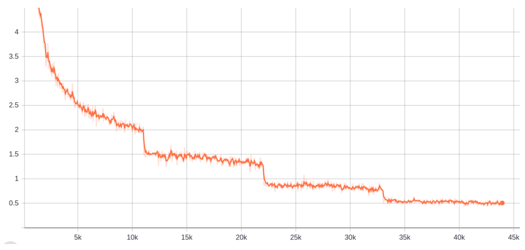


(a) General Loss trend during training

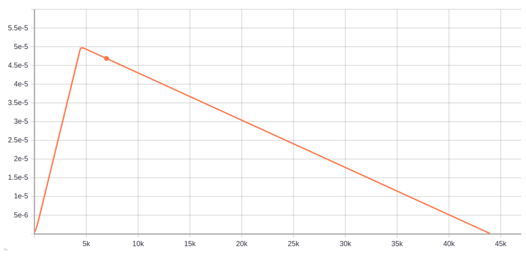


(b) learning rate trend during training

Figure C.1: training hyperparameters: learning rate: $3e - 5$, batch size: 12, epochs number: 3, warmup period: 10%

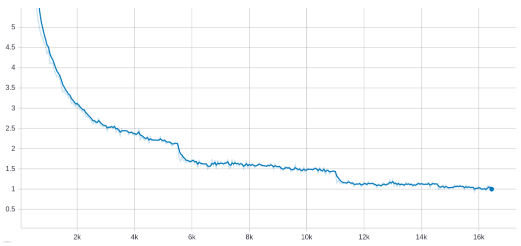


(a) General Loss trend during training

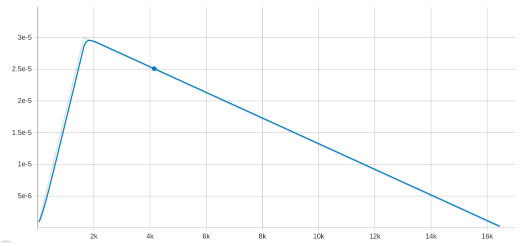


(b) learning rate trend during training

Figure C.2: training hyperparameters: learning rate: $5e - 5$, batch size: 12, epochs number: 4, warmup period: 10%

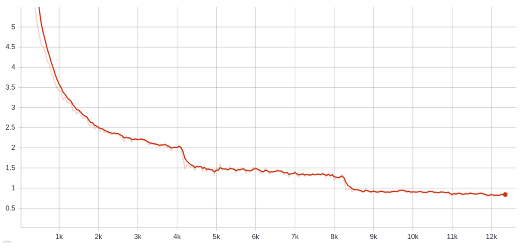


(a) General Loss trend during training

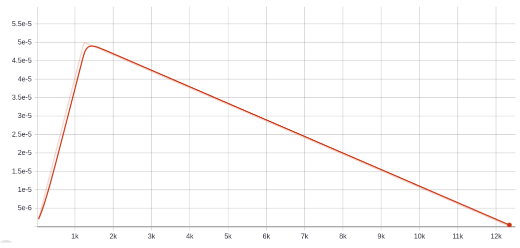


(b) learning rate trend during training

Figure C.3: training hyperparameters: learning rate: $3e-5$, batch size: 24, epochs number: 3, warmup period: 10%

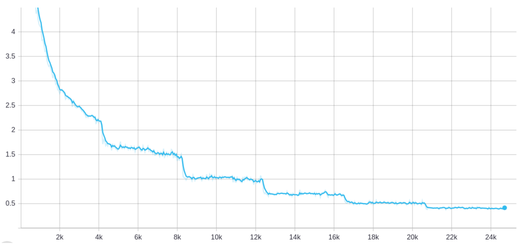


(a) General Loss trend during training

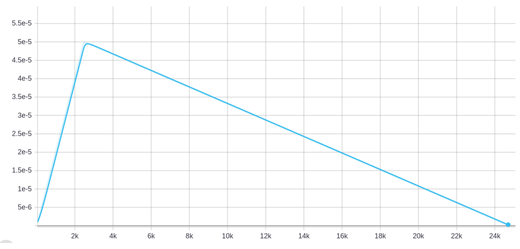


(b) learning rate trend during training

Figure C.4: training hyperparameters: learning rate: $5e-5$, batch size: 32, epochs number: 3, warmup period: 10%

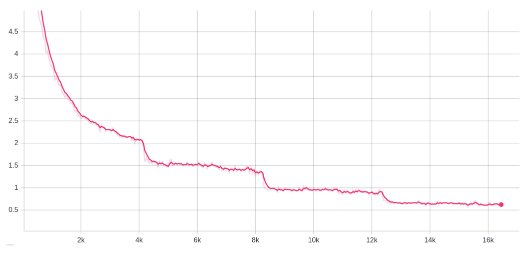


(a) General Loss trend during training

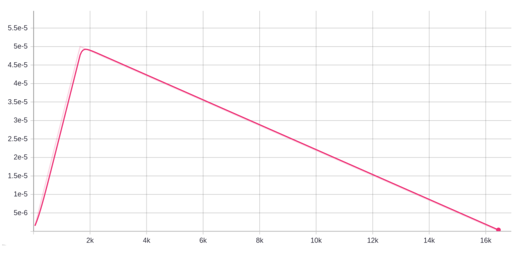


(b) learning rate trend during training

Figure C.5: training hyperparameters: learning rate: $5e-5$, batch size: 32, epochs number: 6, warmup period: 10%



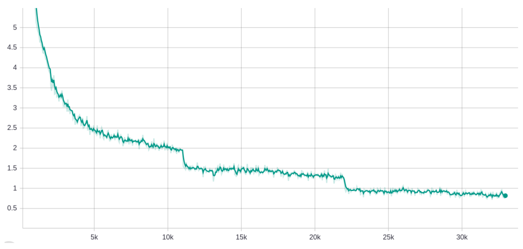
(a) General Loss trend during training



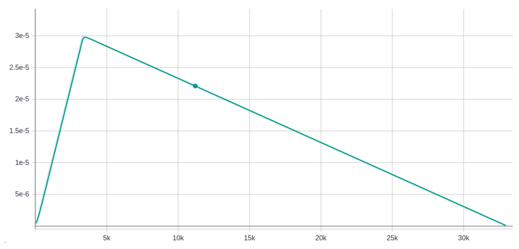
(b) learning rate trend during training

Figure C.6: training hyperparameters: learning rate: $5e - 5$, batch size: 32, epochs number: 4, warmup period: 10%

C.3 QADistilBERT4LGELUSkip

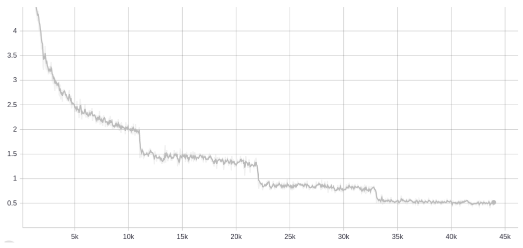


(a) General Loss trend during training

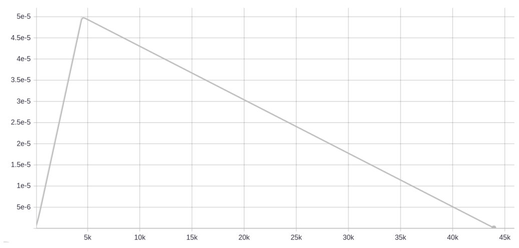


(b) learning rate trend during training

Figure C.7: training hyperparameters: learning rate: $3e-5$, batch size: 12, epochs number: 3, warmup period: 10%

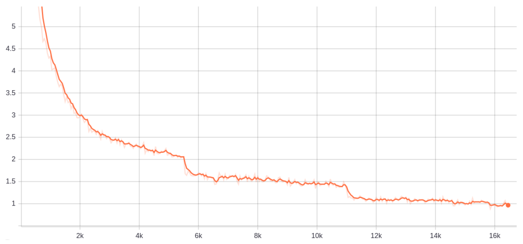


(a) General Loss trend during training

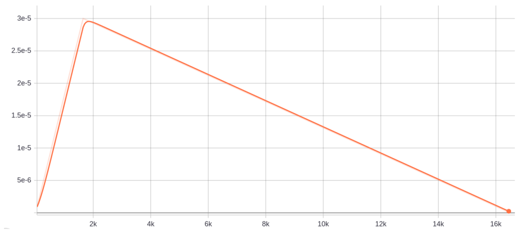


(b) learning rate trend during training

Figure C.8: training hyperparameters: learning rate: $5e-5$, batch size: 12, epochs number: 4, warmup period: 10%

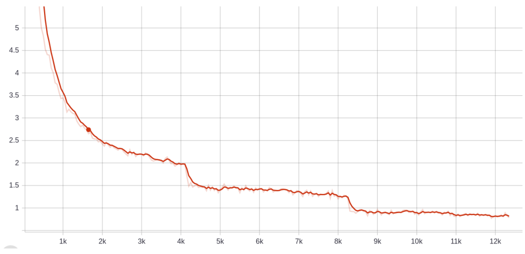


(a) General Loss trend during training

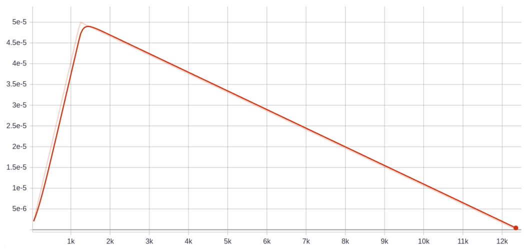


(b) learning rate trend during training

Figure C.9: training hyperparameters: learning rate: $3e-5$, batch size: 24, epochs number: 3, warmup period: 10%

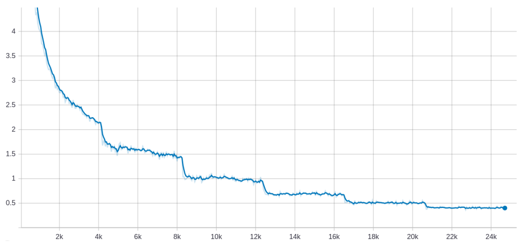


(a) General Loss trend during training

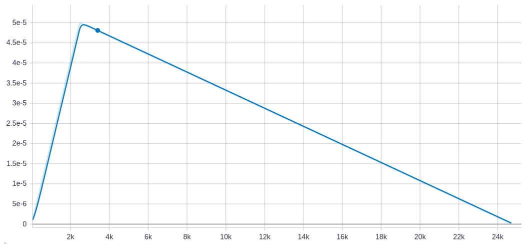


(b) learning rate trend during training

Figure C.10: training hyperparameters: learning rate: $5e-5$, batch size: 32, epochs number: 3, warmup period: 10%

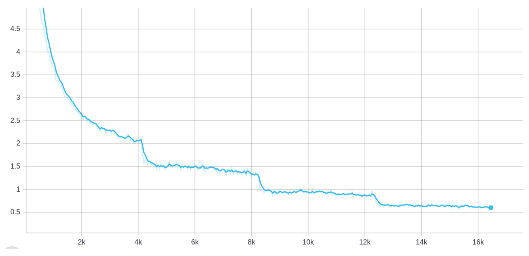


(a) General Loss trend during training

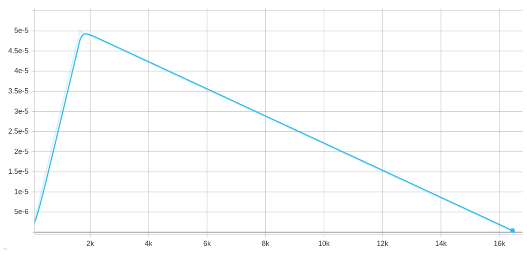


(b) learning rate trend during training

Figure C.11: training hyperparameters: learning rate: $5e-5$, batch size: 32, epochs number: 6, warmup period: 10%



(a) General Loss trend during training



(b) learning rate trend during training

Figure C.12: training hyperparameters: learning rate: $5e-5$, batch size: 32, epochs number: 4, warmup period: 10%

Appendix D

Question Answering Models

D.1 QABERT4LGELUSkip

Listing D.1: QABERT4LGELUSkip

```
class QABERT4LGELUSkip(BertPreTrainedModel):

    def __init__(self, config):
        super(QABERT4LGELUSkip, self).__init__(config)
        self.num_labels = config.num_labels

        self.bert = BertModel(config)
        self.middleOut1 = nn.Linear(config.hidden_size,
           1024)
        self.middleOut2 = nn.Linear(1024, 768)
        self.middleOut3 = nn.Linear(768, 384)
        self.qa_outputs = nn.Linear(384, config.
           num_labels)
        assert config.num_labels == 2
        self.dropout = nn.Dropout(config.
           hidden_dropout_prob)
        self.init_weights()

    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        start_positions=None,
        end_positions=None,
    ):

        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
        )

        sequence_output = outputs[0]
        midOut1 = self.dropout(gelu_new(self.middleOut1(
           sequence_output)))
        midOut2 = self.dropout(gelu_new(self.middleOut2(
           midOut1)))
        midOut3 = self.dropout(gelu_new(self.middleOut3(
           midOut2 + sequence_output)))


        logits = self.qa_outputs(midOut3)
        start_logits, end_logits = logits.split(1, dim
           =-1)
        start_logits = start_logits.squeeze(-1)
        end_logits = end_logits.squeeze(-1)

        outputs = (start_logits, end_logits,) + outputs
           [2:]
        if start_positions is not None and end_positions
            is not None:
            if len(start_positions.size()) > 1:
                start_positions = start_positions.
                   squeeze(-1)
            if len(end_positions.size()) > 1:
                end_positions = end_positions.squeeze
                   (-1)
            ignored_index = start_logits.size(1)
            start_positions.clamp_(0, ignored_index)
            end_positions.clamp_(0, ignored_index)

            loss_fct = CrossEntropyLoss(ignore_index=
               ignored_index)
            start_loss = loss_fct(start_logits,
               start_positions)
            end_loss = loss_fct(end_logits,
               end_positions)
            total_loss = (start_loss + end_loss) / 2
            outputs = (total_loss,) + outputs

        return outputs  # (loss), start_logits,
           end_logits, (hidden_states), (attentions)
```

D.2 QADistilBERT4LGELUSkip

Listing D.2: QADistilBERT4LGEkLUSkip

```
class QADistilBert4LGELUSkip(DistilBertPreTrainedModel):

    def __init__(self, config):
        super(QADistilBert4LGELUSkip, self).__init__(
           config)

        self.distilbert = DistilBertModel(config)
        self.middleOut1 = nn.Linear(config.dim, 1024)
        self.middleOut2 = nn.Linear(1024, 768)
        self.middleOut3 = nn.Linear(768, 384)
        self.qa_outputs = nn.Linear(384, config.
           num_labels)
        assert config.num_labels == 2
        self.dropout = nn.Dropout(config.qa_dropout)

        self.init_weights()

    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        head_mask=None,
        inputs_embeds=None,
        start_positions=None,
        end_positions=None,
    ):

        distilbert_output = self.distilbert(
            input_ids=input_ids, attention_mask=
               attention_mask, head_mask=head_mask,
               inputs_embeds=inputs_embeds
        )
        hidden_states = distilbert_output[0]  # (bs,
           max_query_len, dim)
        midOut1 = self.dropout(gelu(self.middleOut1(
           hidden_states)))
        midOut2 = self.dropout(gelu(self.middleOut2(
           midOut1)))
        midOut3 = self.dropout(gelu(self.middleOut3(
           midOut2 + hidden_states)))


        logits = self.qa_outputs(midOut3)  # (bs,
           max_query_len, 2)
        start_logits, end_logits = logits.split(1, dim
           =-1)
        start_logits = start_logits.squeeze(-1)  # (bs,
           max_query_len)
        end_logits = end_logits.squeeze(-1)  # (bs,
           max_query_len)

        outputs = (start_logits, end_logits,) +
           distilbert_output[1:]
        if start_positions is not None and end_positions
            is not None:
            if len(start_positions.size()) > 1:
                start_positions = start_positions.
                   squeeze(-1)
            if len(end_positions.size()) > 1:
                end_positions = end_positions.squeeze
                   (-1)
            ignored_index = start_logits.size(1)
            start_positions.clamp_(0, ignored_index)
            end_positions.clamp_(0, ignored_index)

            loss_fct = nn.CrossEntropyLoss(ignore_index=
               ignored_index)
            start_loss = loss_fct(start_logits,
               start_positions)
            end_loss = loss_fct(end_logits,
               end_positions)
            total_loss = (start_loss + end_loss) / 2
            outputs = (total_loss,) + outputs

        return outputs  # (loss), start_logits,
           end_logits, (hidden_states), (attentions)
```

Bibliography

[1] https : / / www . socialibreria . com / index . php / 2019 / 10 / 05 / comprensione-della-necessita-di-pnl-nel-tuo-chatbot/.

[2] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. 2018. arXiv: 1803.08375 [cs.NE].

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[4] G. Bebis and M. Georgiopoulos. "Feed-forward neural networks". In: *IEEE Potentials* 13.4 (1994), pp. 27–31. ISSN: 1558-1772. DOI: 10 . 1109/45.329294.

[5] Paul Buitelaar, Philipp Cimiano, and Berenike Loos, eds. *Proceedings of the 2nd Workshop on Ontology Learning and Population: Bridging the Gap between Text and Knowledge*. Sydney, Australia: Association for Computational Linguistics, July 2006. URL: https://www.aclweb.org/anthology/W06-0500.

[6] Paul Buitelaar et al. "Ontology-Based Information Extraction and Integration from Heterogeneous Data Sources". In: *Int. J. Hum.-Comput. Stud.* 66.11 (Nov. 2008), 759–788. ISSN: 1071-5819. DOI: 10.1016/j.ijhcs.2008.07.007. URL: https://doi.org/10.1016/j.ijhcs.2008.07.007.

[7] *CC-NEWS*. http://web.archive.org/web/20200113230124/http://commoncrawl.org/2016/10/news-dataset-available/.

[8] Kyunghyun Cho et al. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches.* 2014. arXiv: 1409.1259 [cs.CL].

[9] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[10] Pranay Dugar. *Transformer - Attention is all you need.* https://towardsdatascience.com/transformer-attention-is-all-you-need-1e455701fdd9.

[11] Aaron Gokaslan and Vanya Cohen. *OpenWebText Corpus.* http://Skylion007.github.io/OpenWebTextCorpus. 2019.

[12] Huggingface Group. *Huggingface homepage.* https://huggingface.co/.

[13] Song Han et al. *DSD: Dense-Sparse-Dense Training for Deep Neural Networks.* 2016. arXiv: 1607.04381 [cs.CV].

[14] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs).* 2016. arXiv: 1606.08415 [cs.LG].

[15] Andreas Herz et al. "The Hebb Rule: Storing Static and Dynamic Objects in an Associative Neural Network". In: *EPL (Europhysics Letters)* 7 (July 2007), p. 663. DOI: 10.1209/0295-5075/7/7/016.

[16] R. Higashinaka et al. "Question answering technology for pinpointing answers to a wide range of questions". In: 11 (July 2013).

[17] Rani Horev. *BERT Explained: State of the art language model for NLP.* https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

[18] keitakurita. *An Overview of Normalization Methods in Deep Learning.* https://mlexplained.com/2018/11/30/an-overview-of-normalization-methods-in-deep-learning/.

[19] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. arXiv: 1412.6980 [cs.LG].

[20] *Knowledge Distillation.* https://nervanasystems.github.io/distiller/knowledge_distillation.html.

[21] Ravindra Kompella. https://towardsdatascience.com/knowledge-distillation-and-the-concept-of-dark-knowledge-8b7aed8014ac. 2018.

[22] Taku Kudo and John Richardson. "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.* Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 66–71. DOI: 10.18653/v1/D18-2012. URL: https://www.aclweb.org/anthology/D18-2012.

[23] Zhenzhong Lan et al. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations.* 2019. arXiv: 1909.11942 [cs.CL].

[24] Yann LeCun et al. "Handwritten Digit Recognition with a Back-Propagation Network". In: *Advances in Neural Information Processing Systems 2.* Ed. by D. S. Touretzky. Morgan-Kaufmann, 1990, pp. 396–404. URL: http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf.

[25] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* 2019. arXiv: 1907.11692 [cs.CL].

[26] Tomas Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality.* 2013. arXiv: 1310.4546 [cs.CL].

[27] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space.* 2013. arXiv: 1301.3781 [cs.CL].

[28] Matthew E. Peters et al. *Deep contextualized word representations.* 2018. arXiv: 1802.05365 [cs.CL].

[29] Pranav Rajpurkar, Robin Jia, and Percy Liang. "Know What You Don't Know: Unanswerable Questions for SQuAD". In: *CoRR* abs/1806.03822 (2018). arXiv: 1806.03822. URL: http://arxiv.org/abs/1806.03822.

[30] Pranav Rajpurkar et al. "SQuAD: 100, 000+ Questions for Machine Comprehension of Text". In: *CoRR* abs/1606.05250 (2016). arXiv: 1606.05250. URL: http://arxiv.org/abs/1606.05250.

[31] Pranav Rajpurkar et al. *SQuAD homepage*. https://rajpurkar.github.io/SQuAD-explorer/.

[32] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

[33] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2019. arXiv: 1910.01108 [cs.CL].

[34] Min Joon Seo et al. "Bidirectional Attention Flow for Machine Comprehension". In: *CoRR* abs/1611.01603 (2016). arXiv: 1611.01603. URL: http://arxiv.org/abs/1611.01603.

[35] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306. ISSN: 0167-2789. DOI: 10.1016/j.physd.2019.132306. URL: http://dx.doi.org/10.1016/j.physd.2019.132306.

[36] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[37] Ralf C. Staudemeyer and Eric Rothstein Morris. *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*. 2019. arXiv: 1909.09586 [cs.NE].

[38] Trieu H. Trinh and Quoc V. Le. *A Simple Method for Commonsense Reasoning*. 2018. arXiv: 1806.02847 [cs.AI].

[39] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[40] Mahendran Venkatachalam. *Self Attention and Transformers*. https://towardsdatascience.com/self-attention-and-transformers-882e9de5edda.

[41] Yonghui Wu et al. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. arXiv: 1609.08144 [cs.CL].

[42] Chen Xing et al. *A Walk with SGD*. 2018. arXiv: 1802.08770 [stat.ML].

[43] Zhilin Yang et al. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. 2019. arXiv: 1906.08237 [cs.CL].

[44] Yukun Zhu et al. *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*. 2015. arXiv: 1506.06724 [cs.CV].

Acknowledgements

I wish to express my deepest gratitude to my supervisors, Prof. Dr. Philipp Cimiano and Prof. Dr. Fabio Tamburini, for their availability during the research period, both in Bielefeld and Bologna.

I would like to pay a special mention to Frank Grimm, which was always ready to give answers and comparisons.

There is no way to express the importance of the collaboration with Matteo Del Vecchio, whit whom not only did I share this cooperation, but also my friendship.

I would like to recognize the invaluable support that my family provided me, without pressure, being always on my side during my studies and giving me this precious opportunity for my future. Thanks to my father Silvestro, my mother Ada and my sister and brother, Eleonora and Riccardo.

I would particularly thank my girlfriend, Laura for being patient and supporting me even in the worst periods, and of course for revising all this document.

To my friends Giulia, Anna and Matteo for being my second family during our experience abroad, for all the moments we shared.

Finally, I would like to thank all of my friends, that I did not name in these few rows above, for being part of my life.