Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

Optimization of State of the Art Techniques for Natural Language Understanding

Autor: Ariel Alejandro Fabiano
Tutor: Alfonso Mateos Caballero

Madrid, Marzo 2020

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial

Título: Optimization of State of the Art Techniques for Natural Language Understanding

Marzo 2020

Autor: Ariel Alejandro Fabiano
Tutor: Alfonso Mateos Caballero
Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Abstract

One of the most complex challenges of Artificial Intelligence is to fully understand language the same way humans do. Natural Language Understanding is a field that has had some of the most groundbreaking innovations in the last years, yet is still far from having cognitive capabilities. The following work covers parts of the evolution of this field and uses a novel challenge proposed by Google as the problem to test how well these theories hold up to the task. Several techniques are tested up to state of the art standard and for the top performing model an extensive fine-tuning process is performed. As part of a competition held by Kaggle, the goal of this work will not only be a scientific study of these models, but a log of the process involving the creation of a model capable of achieving top performance.

Contents

Chapter 1

Introduction

This chapter introduces the objective of this thesis and covers an overview of the general aspects related to the work performed in it and the considerations to take into account.

1.1 Problem definition

The goal of this work will be to explore the effectiveness of several existing Natural Language Processing (NLP) techniques on a non-traditional question-answering task. The problem itself was proposed by Google on a Kaggle competition [1] and it was defined as a subjective question-answering labeling challenge.

Traditional question-answering (Q&A) systems have been one of the principal tasks when trying to benchmark Natural Language Understanding (NLU) algorithms [2] and they tend to focus on fact seeking information through text corpora in order to answer a specific question or judge a proposed answer.

In contrast, for the problem at hand the goal changes as the idea is to be able to understand subjective aspects of the question and answer. These type of characteristics obtained from the text are based on cognitive abilities a human can employ while analyzing a piece of text and, so far, machines are still unable to comprehend.

1.2 Background

When trying to address questions that require a subjective analysis, such as asking for recommendations or opinion seeking, there is a current lack of computer methods capable to interpret the real meaning of what is being asked and retrieve or create a meaningful answer with information that exceeds of what can be found in plain text.

Although some efforts have been made to improve Q&A systems with ontologies from specific domains [3] or with information represented through Knowledge Graphs [4], most systems focused on the task of parsing the question in order to expand the answer through upper knowledge obtained from the additional database. While being able to fully incorporate general knowledge databases in a semantic and cognitive way will likely be the final form of Q&A systems, we are still far from a system that can interpret a question on a human level.

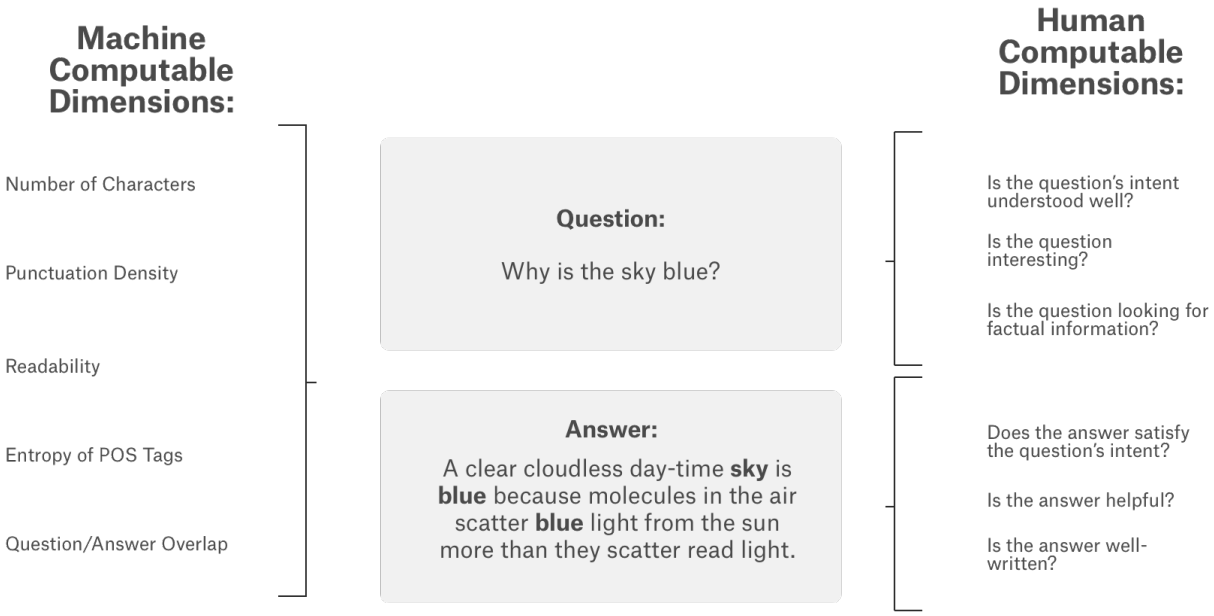


Figure 1.1: Possible concepts for interpretation of Q&A [1]

In order to explore the possibilities of expanding machine computable dimensions of language understanding, Google presented a novel dataset where they created 30 handcrafted labels for 6079 pairs of questions and answers collected from nearly 70 different websites. These Q&A where rated for each of the 30 labels in a "common-sense fashion" by a group of raters without a proper guideline, forcing them to rely on their intuition for the subjective interpretation of the prompt and how would it be valued.

The purpose of this dataset is to test if existing methods are capable to predict properly these subjective labels and in the process demonstrate that there are ways to detect cognitive dimensions from questions and answers that vary in content and form, such as: multi-sentence elaboration, advice or opinion seeking, interesting topics, right or wrong answers, and more.

1.3 Motivation and Purpose

Considering the unique nature of the labels to predict from this dataset, a valuable research would come from understanding the readiness of existing Natural Language Processing (NLP) techniques to thrive in such task. In particular, it would be interesting to see the behaviour of the current NLP's state of the art star: Bidirectional Encoder Representations from Transformers (BERT) [5] and how much it can be fine-tuned to improve its performance.

BERT has been considered as a game changer in the world on NLP, creating new opportunities on how language representation can be used [6] and improving by at least 4.6% the state of the art performance on most of the traditional language benchmarks. The success of this new model has encouraged several works to focus on how to optimize BERT for classification [7] or how to boost its performance by adding enriched information [8], all of them obtaining successful results in traditional Q&A problems.

The purpose of this thesis will be, initially, to study the performance on NLU for a set of NLP algorithms evolving in complexity, trying to specify what they are lacking to obtain a better result. Lastly, the focus will shift into iterating over different configurations of the best performing algorithm, trying to calibrate it to achieve the best result.

Once this work is completed, it is expected to have a better understanding of the challenges involving subjective Q&A and how valuable Google's new dataset is for the NLU field. Also, through this process, several proposed optimization techniques will be validated empirically and a deeper knowledge of how they interact between each other could be asserted.

1.4 Scope and Limitation

Apart of the NLP models and their capabilities, there is a need to explore the composition of the dataset to understand better the challenge and its potential problems. The dataset published by Google [9], has 10 trainable features, that can be described in the following way:

- *question_title:* free text consisting on the title of the question
- *question_body:* free text consisting on the body (i.e. explanation) of the question
- *answer:* free text consisting on the answer to the question
- *host:* categorical feature containing the address to 63 different StackExchange sites
- *category:* categorical feature with 5 possible values related to the question topic
- *question_user_name:* name of the user that asked the question
- *question_user_page:* URL of the profile the user that asked the question
- *answer_user_name:* name of the user that answered the question
- *answer_user_page:* URL of the profile of the user that answered the question
- *url:* URL to the question web page

When considering the real value that they can offer for a model that seeks to understand subjective dimensions of text, only the first 5 could be of use. In the case of *host* and *category* they could provide valid context that can relate to the structure and topic of the text (e.g. StackOverflow questions can introduce code segments which are of a different nature than regular text).

	Question Title Char Length	Question Title Word Length	Question Body Char Length	Question Body Word Length	Answer Char Length	Answer Title Word Length
mean	53.31	10.14	833.76	174.01	839.40	175.62
std	20.21	4.59	1029.05	204.44	1017.39	229.63
min	15.00	2.00	1.00	0.00	21.00	4.00
25%	39.00	7.00	323.00	66.00	297.00	60.00
50%	50.00	9.00	544.00	115.00	556.00	115.00
75%	65.00	13.00	969.50	204.00	1015.50	212.00
max	147.00	47.00	19253.00	3077.00	22636.00	8184.00

Figure 1.2: Free text features length detail

The main features of this dataset are exclusively related to free text fields and they greatly fluctuate in length as detailed in Figure 1.2. Based on the fact that there are only 6079 training samples, the variance on text structure for the Q&A pairs might prove challenging for basic NLP methods and could even have a negative impact on newer techniques that work with fixed maximum length.

Completing the dataset are the labels to predict, which are composed by decimal numbers in the range of 0 to 1, and they are the result of averaging the score given by several raters for each label. The distribution of the values for these 30 labels can be seen in Figure 1.3 and it shows that at least a third of them are unbalanced (e.g. *question_not_really_a_question* has almost all it values set to 0).

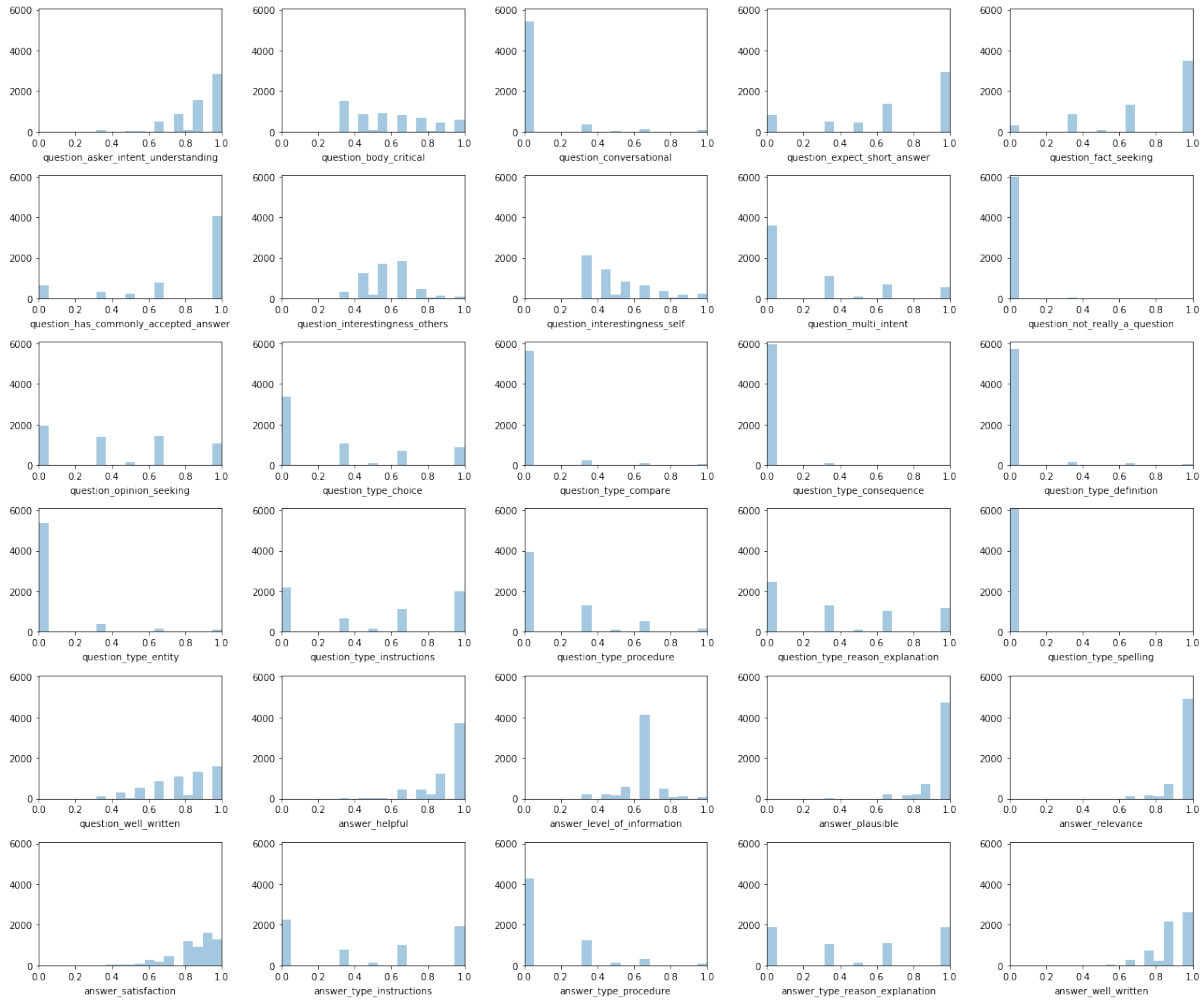


Figure 1.3: Labels value distribution

For this specific competition Google decided that the best metric to predict was Spearman rank correlation coefficient (ρ), which would produce a ranking of the best to worst examples of text that associate to each label. The overall performance metric is measured by calculating the Spearman's ρ for each column and then averaging all values together. The idea behind this is to obtain a model that can interpret multiple subjective dimensions and order them independently.

Due to this problem being part of a Kaggle competition, there are some rules that will

translate into limitations for this research. Initially, for baseline models exploration, there will be a time limitation on the training and inference process set to a maximum of 2 hours, which will encourage clean, simple solutions, without an excess of validation techniques. Once the best performing model is found, further improvements will be explored with the same rules, and finally the best performing model will be trained without a time constraint, but with an inference time limit of 2 hours.

Another limitation is the fact that there is no real access to the test datasets, being only possible to obtain the resulting Spearman ρ average score for them. As Kaggle tries to avoid test dataset probing (i.e. learning about the real values to predict by trial and error) in their competitions, they also limit the quantity of submissions to 5 by day, which reinforces the idea of finding the best performing model as fast as possible and then proceed to perform experiments on its optimization in a contained manner. Is for this reason that in this thesis the metrics will have a different level of detail for the training/validation process compared with the final test score.

1.5 Outline

The rest of this document is structured in 6 more chapters, consisting of:

- Chapter 2 covers the *theoretical background* needed to understand the NLP techniques used in this work and focuses on the state of the art of NLU, mainly the BERT model and its foundations.
- Chapter 3 describes the *development environment* used to train the models and the *research method* used to validate them.
- Chapter 4 details the *results* of the baseline models experimentation and decides the candidate to be optimized.
- Chapter 5 shows the *fine-tuning* process performed to create a model capable of winning a medal in the competition.
- Chapter 6 provides some final thoughts and gives a *conclusion* to the work performed on this thesis.

Chapter 2

Theoretical Background

The following chapter covers basic concepts of NLP, provides an overview of how parts of the field evolved through time with more complex techniques and introduces Deep Learning concepts used during this work.

2.1 Natural Language Processing

NLP is one of the main fields of study of Computational Linguistics and it has a deep connection with Computer Science and Artificial Intelligence. Its focus is on how machines can interact with human language and it has been used as cornerstone for search engines, speech recognition, language translation and many more.

As part of handling language it employs several specific concepts, some of which can be summarized in the following manner:

- *Corpus:* a collection of text documents considered the main input for most NLP tasks.
- *Lexicon:* one of the main linguistic resources that contain a dictionary of verbs, adverbs, adjectives and substantives used for semantic interpretation (e.g. WordNet [10]).
- *Knowledge Base:* capable of storing unstructured and structured information and key element for organizing concepts extracted from text (e.g. taxonomies, ontologies).
- *Information Extraction:* process of automatically extracting structured information from an unstructured source (such as a text corpus).
- *Information Retrieval:* obtains relevant information from a corpus of text based on a specific criteria.
- *Named Entity Recognition:* it is a specific case of information extraction where the goal is to detect a named entity that fit into particular categories (i.e.: person names, organizations, etc.).

As a general guide in an NLP project, it is expected to find preprocessing techniques to perform text clean-up and standardization, which will heavily relay in syntax process-

ing, in order to later proceed with feature engineering through Information Retrieval (IR) or language modeling techniques.

2.1.1 Information Extraction

Languages contain a set of syntactic rules on how words have to be arranged in order to make grammatical sense, and these grammatical rules are the ones used in NLP in order to process text to prepare it for Information Extraction (IE). One of the most famous tools used to apply this process is the NLTK library [11], maintained by Princeton University and reference for several IE and IR techniques.

2.1.1.1 Text cleanup

Prior to diving into syntax processing, it is important to note that with web text (which likely contain parts that do not respect syntax rules) it is always better to perform a set of text cleanup strategies such as: removal of HTML tags, extra whitespaces and special characters, conversion of number words to numeric form and accented characters to ASCII, validation of letter case (specific to the use case, but conversion to lowercase is the general practice) and, in case the language allows it, expansion of used contractions (e.g. isn't to is not).

2.1.1.2 Tokenization

Parsing text, or text chunking, is the initial step into applying most of NLP techniques. The main idea is to transform the text into a list of tokens, where the splits can be performed by paragraph, sentence, word, word stem or even custom splitting logic. This array of tokens will allow faster computation by creating a representation of the text closer to a numerical form (i.e. replacing all words by a set of ids).

2.1.1.3 Stopwords

It is said that Hans Peter Luhn, pioneer in the field of IR, coined the concept of this list of specific words that have a high frequency in a language [12]. A common practice is to remove this words as they are considered to not carry any real information through the text.

In general cases, the stopword list from the NLTK library offers a standard collection of frequent words that, in most cases, is better removing. These words are usually articles, connectors and prepositions, and the list can be customized for specific domains where it might be better to keep them or add domain related stopwords.

2.1.1.4 Part-of-Speech Tagging

Part of Speech (POS) is a type of word categorization that classifies the grammatical syntactic behaviour. Tagging all words in a text by their POS composition can help to describe better its contents and structure, main reason why it is a common strategy in NLP.

The main tag set in the English language is considered to be the TreeBank (part of the NLTK library), which considers that there are eight main POS: nouns, pronouns, adjectives, verbs, adverbs, prepositions, conjunctions and interjections [13].

Also TreeBank utilizes 4 more extra POS: cardinal numbers, other/foreign words, punctuation and particles.

2.1.1.5 n-Grams

Considering text as a sequence of words (tokens), a unigram would be composed by a single token (e.g. Buenos), whereas a bigram would the selection of two adjacent tokens (e.g. Buenos Aires). This concept allows to navigate through text using a sliding window which will group tokens and allow to perform calculation of how words relate between each other.

An empirical way to understand the usefulness of this is the n-Gram model, which will calculate the probability of occurrence of words depending on their context. For example, the sentence "In the morning I drink" has a high probability of being followed by the words "tea" or "coffee", options that could be obtained by previously analyzing several corpora with a 4-Gram stride.

2.1.1.6 Stemming and Lemmatization

Most languages are composed by an initial set of words that are later expanded (e.g. plurals), conjugated (e.g. verbs), mixed (e.g. compound words) and more. In linguistics, stemming and lemmatization are a way to reduce this inflectional forms and to return words into their base form [14].

There are different algorithms capable of performing this task and their difference is based on how they detect the boundaries of the word to transform, nevertheless the main idea remains the same in all cases, whereas stemming will perform a basic heuristic removing the end of a word after a detected stem and lemmatization will analyze the morphology or vocabulary association of the word and try to detect its lemma (i.e. base form of the word).

These techniques allow for a better compression of the information present in the text, reducing the variation of language and generating bigger associations of the words inside their context.

2.1.2 Information Retrieval

One of the primary tasks of IR is text indexation, which initially started as a technique focused on search engines, but has greatly contributed in several IR statistical strategies and helped to develop valuable relevance metrics used for feature engineering. Is in this scenario where calculating the importance of particular terms in a corpus helps creating new features useful for Machine Learning (ML).

2.1.2.1 Bag of Words

A basic method of representing text data is to count the appearance of each word in a document and keeping track of it in a dictionary, where each word will have a frequency associated to it. This technique is called Bag of Words (BoW) and is considered a context-free language representation.

BoW will allow the creation of an array with the raw count of terms (or frequency in cased it is pondered by the total quantity of unique terms), and produces a simple

way of indexation for searching words inside documents. In the case of ML models this technique resembles the concept used in One Hot Encoding (OHE) for categorical features, and is usually named BoW model encoding.

2.1.2.2 Term Frequency-Inverse Document Frequency

Another context-free language representation and an improved method over BoW to help indexation by also giving an importance metric of the association of the term in the document. In [14] Term Frequency-Inverse Document Frequency (TF-IDF) is described as the combination of TF (i.e. frequency of a word through a document) and IDF (i.e. inverse frequency of the word through all documents). This combination produces a score where relevant words for the document, and not for the full set of documents, will have a score close to 1 and words frequent in all documents will tend to 0.

The formula that describes classic TF-IDF (with division by zero adjustment) is the following:

$$tf - idf_{t,d} = f_{t,d} \Big/ \sum_{t' \in d} f_{t',d} \cdot \log \frac{N}{1 + |\{d \in D : t \in d\}|}$$

with term t, specific document d, document collection D, and N as quantity of documents.

A common practice of text representation for prediction models involves selecting the top k TF-IDF scoring words in all the documents and creating k features, assigning the value related to word in each document. This could be even further improved by selecting all words present, mapping their TF-IDF values to each document and then performing a dimensionality reduction process (e.g. Singular Value Decomposition (SVD)) to have a more manageable and less sparse (i.e. compressed) set of features [15]. This technique could be considered a basic form of word embeddings.

2.2 Deep Learning

A specific field on the family of classes of ML and one of the most groundbreaking areas of study in the current time, Deep Learning (DL) bases its success in differential programming and the subsequent optimization of several algebraic techniques used in multi layered Artificial Neural Networks (ANN) [16].

A basic form of ANN can be considered the Multi Layer Perceptron (MLP) composed by an input layer (dataset features), a unique hidden layer that combines the inputs (activation function) and an output layer (label to predict). This will be then trained through several epochs or subsets of the dataset (i.e. mini-batches) trying to minimize the loss between the output and the real value through a specific loss function formula and using an optimizer technique that allows searching better differentiable solutions and adjusting the parameters through backpropagation.

2.2.1 Activation function

While combining input nodes across layers in ANN there are several methods to obtain the resulting output, these functions are a set of mathematical equations capable

of determining when a node has to be activated or not based on its relevance to minimize the loss.

The principle works in a way that allows linear and non-linear functions, meaning that input data can be combined to obtain more complex relations and deeper knowledge can be found. With that consideration and the objective of optimizing the parameters of network, one aspect to consider is the derivative function associated to the activation method as it will be crucial for gradient optimization.

Basic forms of activation functions are linear (multiplies input nodes by their weights creating an output proportional to the input) and binary step (splits the input by a threshold and outputs 0 or 1), which do not offer the possibility to optimize weights through derivative backpropagation.

2.2.1.1 Rectified Linear Unit

The Rectified Linear Unit (ReLU) is an adaptation of the linear function where all values below 0 will result 0 and the rest will have its linear value. This formula can be seen at Figure 2.1 and thanks to the split output it offers a derivative behaviour of 2 constants (0 and 1), allowing gradient optimization during backpropagation.

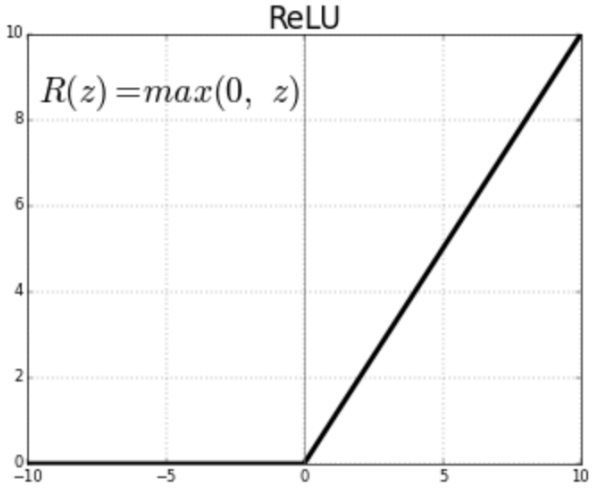


Figure 2.1: ReLU activation function

A problem of this activation function is the dying ReLU, where several of the inputs are negative or close to zero, turning the gradient of the function to zero and preventing the network to adjust its weights.

Several methods were created after the concept of ReLU, trying to solve its deficiencies and also preventing other problems such as gradient vanishing (i.e. multiplication of several low values that decrease gradient learning) or gradient exploding (i.e. some nodes start having increasing weights and gradient learning gets biased only considering those). Some of the most common methods are LeakyReLU, Exponential Linear Unit (ELU) or the newer Scaled Exponential Linear Units (SELU) [17].

2.2.1.2 Sigmoid

This function is often considered as a logistic activation, with an output range between 0 and 1 and a smooth gradient step. It is non-linear in nature and allows to learn through its derivative with normal distribution (with maximum of 0.25 at 0). In the case of binary classification, its output is a good representation of the probability of belonging to the determined positive class.

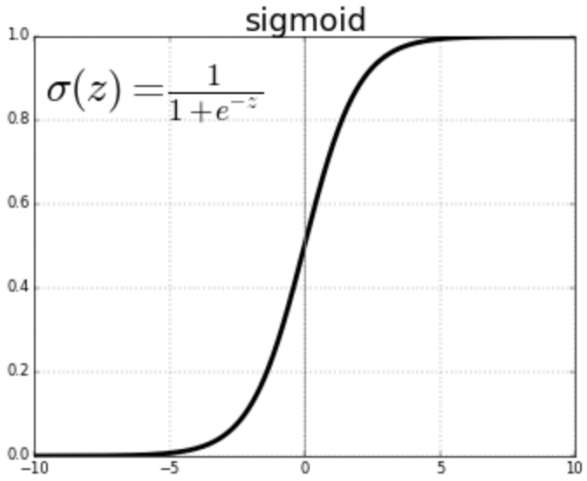


Figure 2.2: Sigmoid activation function

Due to having a tendency to 0 and 1 in its extremes, this can lead to gradients with derivatives close to 0, thus having the problem of vanishing gradient and slowing down how the model converges into its best solution.

Following the family of this function several different formulas can be found, like: TanH (same as Sigmoid but with range between -1 and 1), ArcTan, SoftSign or SoftMax for categorical classification.

2.2.2 Optimizer

The training process of DL models faces the specific challenge of mathematical optimization of an objective function capable of finding the best configuration of parameters to solve the problem at hand.

Several optimization methods have been developed and used in multiple fields, but in the case of ANN the main focus goes to differentiable methods in order to take advantage of the activation functions and backpropagation learning.

2.2.2.1 Stochastic Gradient Descent

It is an algorithm that seeks to reduce the training error by adjusting the ANN node weights iteratively. Mini-batch Stochastic Gradient Descent (SGD) is an improved version of traditional gradient descent, with the main difference being that the learning process will be performed in small steps by splitting the dataset in batches and then picking observations randomly.

Through this process, the error (i.e. difference between the predicted and the real value) is calculated at each step and the weights are adjusted using the formula

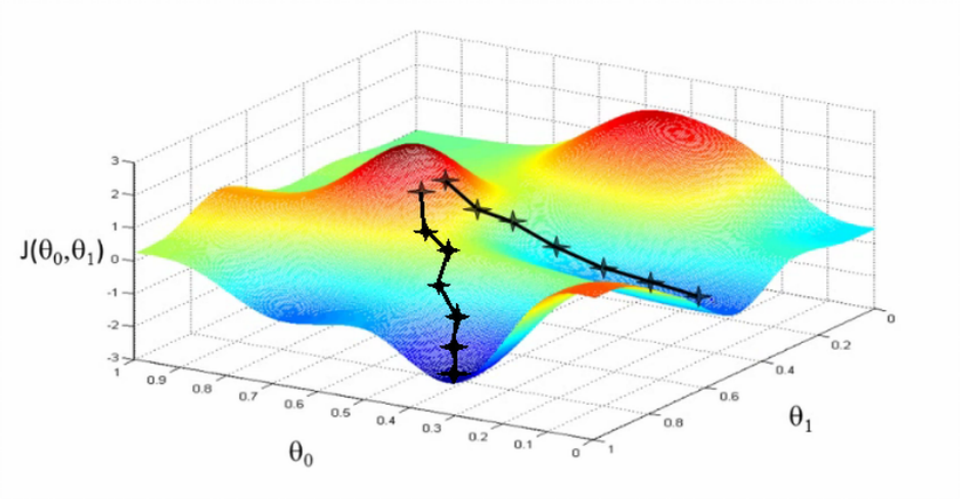


Figure 2.3: Stochastic Gradient Descent optimization steps

defined by the cost function. Repeating this process will make the model slowly converge to a minimum solution where the error reaches its lowest value. Notice in Figure 2.3 the possibility of having a local minimum and global minimum, this is possible due to SGD being a first order gradient based optimization.

One of the main advantages of SGD is that it allows parallelization, which highly speeds up the training process. Another advantage comes from using mini-batches that smooth the learning by decreasing the adjustment oscillation of weights. Also, the convergence of the model can be regulated through the learning rate, a constant value that will scale how much of the adjustment proposed by each step will impact the weights.

An addition to SGD that helped to achieve even better performance and faster training, is Momentum. A simple technique that will take into consideration the previous trajectory of the gradient descent and transfer some of its momentum to the current learnt optimization, preventing any abrupt "zig-zag" in the process of finding the minimum.

2.2.2.2 Adaptive Moment Estimation

Adaptive Moment Estimation (Adam [18]) is currently one of the top optimization techniques. It takes the advantages of SGD with Momentum and combines it with adaptive learning strategies like the ones proposed in Adadelta [19].

Adam is a method that computes adaptive learning rates for each parameter by calculating an exponentially moving average of the gradients (similar to Momentum) and squared gradients (for adaptive learning), while controlling the decay rates of these moving averages.

This combination of methods produces a faster convergence due to a more customized learning process for each parameter (node) in the network. It is important to mention that Adam is still a first order gradient descent optimization technique and can suffer from getting stuck at local minimum and not finding the optimal solution [20].

2.2.3 Loss function

The way to measure how precise an ML model is at the moment of predicting the expected output is through the loss function. The loss is calculated by comparing the real target and the predicted one, and its value comes from measuring this difference through the loss function formula.

Loss functions have intrinsic properties based on the mathematical theory behind their formula. This give each loss function the capacity to perform differently on the same problem, with ones being highly affected by outliers, others being focused only on majority generalization and some specializing in specific target distributions.

2.2.3.1 Mean Squared Error

Mean Squared Error (MSE) will calculate the difference between the predicted value and the ground truth, square it, and average it for the entire dataset. This way the loss will never be negative, helping the backpropagation process.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

MSE offers great performance when the model is working with a dataset without strong outliers and it is the traditional method used for regression problems. Its main drawback comes from the fact that big prediction errors or the instability of outliers will have a heavy impact on the final loss, as its value will be squared.

Another way to calculate the loss can be through Mean Absolute Error (MAE), which is the sum of the absolute value of the difference between prediction and ground truth. This method gives robust results when present with outliers, but fails to predict them if needed.

The two previously mentioned functions cover the disadvantages of the other, and it is why the Huber loss was created. Huber balances both MSE and MAE in its function by using one or the other when the loss exceeds a specified threshold *delta*. This function gives a more rounded performance, obtaining better results with real data.

2.2.3.2 Binary Cross-Entropy

Entropy is a concept used in information theory that, in a way, quantifies the level of information for a given event. This information is often associated with its surprise factor, or their probability of occurrence. It is considered that balanced probability distributions have a high entropy, whereas skewed probability distributions have low entropy.

Cross-Entropy is a way of measuring the difference between the entropy (i.e. probability distributions) of two set of events. It calculates the number of bits (information) required to represent an observation from one distribution compared to another distribution.

When building binary classification models, Binary Cross-Entropy (BCE) provides a robust way to calculate the loss as it will compare the probability distribution of the

positive class in the real dataset against the predicted one. This will measure the distance between the ground truth and what the activation of the nodes in the NN are converging into.

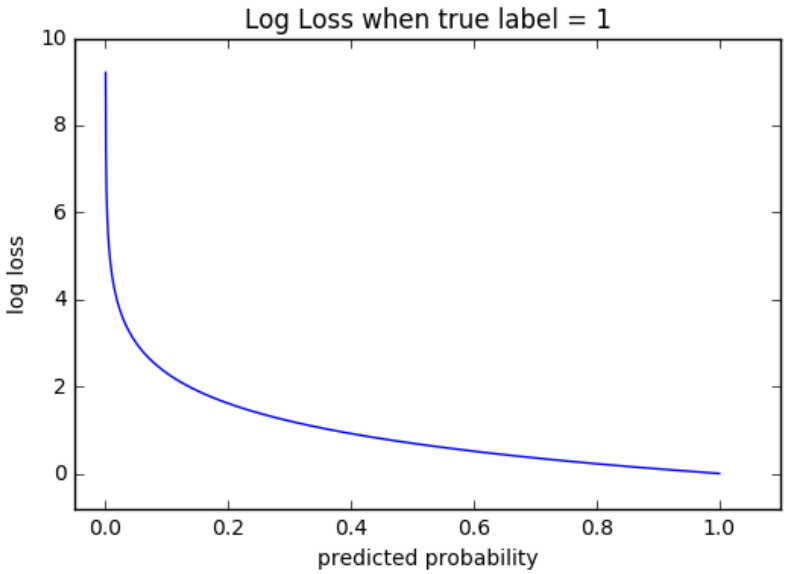


Figure 2.4: Binary Cross-Entropy loss behaviour

In technical terms this loss function will perform a sigmoid (or Softmax for multi class) activation and then calculate the Cross-Entropy loss. Figure 2.4 shows how the log loss (which has the same behaviour as BCE) decreases the closer the predicted probability is to the ground truth.

2.2.4 Artificial Neural Network

Artificial Neural Network models are a connectionist system composed by layers containing a set of nodes (or neurons) where information is combined across the model and their parameters are optimized towards minimizing the loss. The layers will be defined as input layer (one node for each feature of the dataset), the hidden layers (quantity and shape decided while constructing the ANN) and the output layer (containing the target feature o features to predict).

Each node step in between layers will have two basic parameters associated to its nodes. Weights are the coefficients of the equation to optimize and they work in similar fashion as a regression algorithm would do. Bias are constant values that will provide an offset to the result of the product between inputs and weights, helping the result of the activation of the node have less variance and aim for a more generalized solution.

Once the framework has been set (layers, connections between them through the activation functions, parameters initialized), a propagation method will be in charge of passing the observations through the ANN and start creating predictions. This feed forward process computes the value of input neurons as a weighted sum, plus the addition of the bias, and outputs its result after passing through the activation function.

In order to learn and start optimizing, the ANN will measure the error of prediction with the loss function and will adjust the weights proportionally based on the gradient (derivative) of the loss function. This backpropagation method will optimize each connection inside the ANN with a procedure dictated by the optimizer algorithm. In general terms, the optimization process is conceptualized as a search problem with multiple parameters with the goal to reduce the loss, and it is done by adjusting the coefficients depending on how their trajectory (derivative) impact the overall result.

2.2.4.1 Autoencoder

Autoencoders (AE) are a specific architecture design of ANN considered as unsupervised learning. This type of ANN focuses on learning latent representations of the data, usually by performing encoding steps and compressing information into a reduced layer size, to then decode it and try to reconstruct the input data. When just using the output of the encoder (intermediate step) this model can be considered as a dimensionality reduction technique.

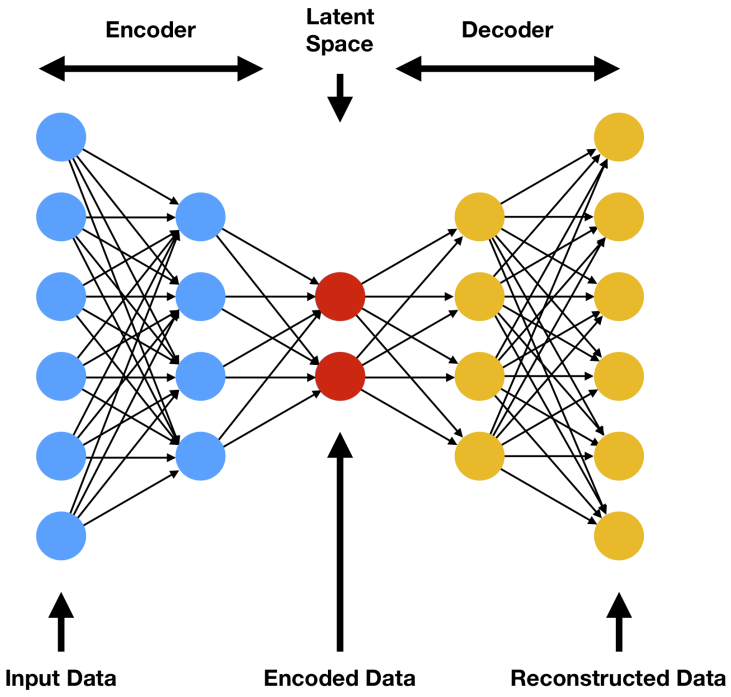


Figure 2.5: Basic Autoencoder Architecture

An AE learns by optimizing its weights through the calculation of the loss at the moment of reconstruction. One of the main advantages of these types of architectures comes from the fact that the knowledge learnt by the encoding step, where a more dense (less sparse) representation of the data has been obtained, can be then reused in prediction problems as the initialization state of a network.

2.2.5 Recurrent Neural Network

This particular family of DL is based on the capacity of process and learn from sequential data (such as speech and language). Recurrent Neural Networks (RNN) use

internal loops in their architecture that allows them to persist previous information from an element in the sequence and use it in favor of understanding the probability of the next element. For example, given the sentence *"New York New York"* an RNN of 3 hidden units (loops) could learn to predict the last word *"York"* given the initial 3.

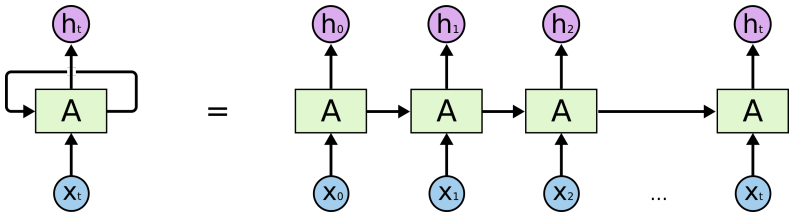


Figure 2.6: Inside functionality of an RNN node

RNN have a modified version of the ANN neuron, capable of maintaining in their hidden units a "state vector" that implicitly contains information about the history of all the past elements of the sequence. This capacity of working with sequences of vectors overcomes the limitation of traditional ANN of only being able to train with fixed length vectors.

In Figure 2.6 it is possible to see how a single node unfolds internally, showing that the loop itself could be thought as a multi step feed forward process storing the activation in each case and passing through the entire sequence while updating weights in the process.

2.2.5.1 Long Short Term Memory

A limitation present on RNN is the fact that it will not be able to hold important initial information on input sequences longer that the length of its hidden units. In order to augment the network with an explicit memory capable of holding long-term dependencies, RNN were improved into Long Short Term Memory (LSTM).

LSTM use a special kind of hidden units that contain an additional memory cell that accumulates valuable information until it stops being considered useful. This can be achieved thanks to the addition of 3 internal interconnected gates in the hidden state that detect the relevance of the information stored in the memory cell based to the loss propagation.

The first gate will be in charge of deciding if the information from the cell state should be forgotten or kept and it is made by a sigmoid layer that will output from 0 (forget) to 1 (keep). Then another gate will decide if the new value has to be stored, also decided by a sigmoid layer and then stored after passing through a TanH layer. Lastly, with the final output for the hidden state has to be calculated, this will be a mix of the basic RNN cell state (from the current information) with the retrieved information stored in memory. This is done by selecting which parts of the cell state to keep through a sigmoid layer and then activating the information with a TanH layer, and then multiplying it with the memory information.

2.2.6 Transfer Learning

Thanks to the nature of ANN of being able to learn internal representations of knowledge, a useful technique can be used to improve performance on specific tasks

through reusing previously trained models.

Transfer learning focuses on expanding the usage of models by offering their pre-trained internal states (architecture, weights, biases) and giving a boost in the warm up process involved while training on a known problem from start. For instance a model previously trained to detect bikes in an image, can be a good starting point to train another model to detect motorbikes.

In general terms, specific and loosely standardized problems (such as language or computer vision) can have a pretrained model with massive information and learn knowledge that can be later reused as input for another model. The process of reusing a model and adjusting it to the new problem at hand (i.e. classify a different label) is called fine-tuning and it has the advantage of offering good results even when the new training data is small.

2.3 Text embeddings

The main goal of a text embedding is to be able to represent text as numbers, which will allow to perform mathematical computations while retaining some of their linguistic meaning. This encoding technique can work on different text levels, going from word, sentence, paragraph up to the whole document.

Although models solely based on word embeddings are not exactly a Neural Language Model (NLM), as they lack the capacity to consider word order or sequence interpretation and can not generate text by themselves, the basic components behind them are the key concepts used to build more complex models on higher text levels and helped understanding specific semantic features through distributional similarity based representations of text.

2.3.1 Word embeddings

This method helps creating a dense representation where similar words have a similar encoding, for example the word *"cat"* and *"dog"* will have a closer distance than *"cat"* and *"car"*. This result can be obtained due to the principle of similar words occurring more frequently together than dissimilar words in text.

Embeddings are encoded dense vectors containing floating point values learned through a training process. The length of the vector is a parameter decided at the moment of the creation of the embedding and it will depend on the amount of data available (longer vectors require more data) and the level of word representation specificity desired (higher dimensions allow for stronger relationship between words).

2.3.1.1 Word2Vec

Word2Vec (W2V) was proposed by Google in 2013 [21] and it triggered the creation of a series of powerful tools that evolved the understanding of semantic meaning in text. Basically an unsupervised deep learning model capable of creating a dense vector representation of words as its output, W2V offers out-of-the-box a set of pre-trained models with different vector lengths, previously trained with massive corpora. The dictionary created by the output of these models can then be used to encode texts and extract vector features containing contextual and semantic information.

W2V can have two different internal architectures that offer different functionalities: Continuous Bag of Words (CBoW) and Skip-gram model, both of which can be used to train a custom dictionary with a given corpora (usually a good practice when trying to work with texts from a specific field). In both cases, the models will create their own labels to predict from the text, transforming the problem into supervised learning.

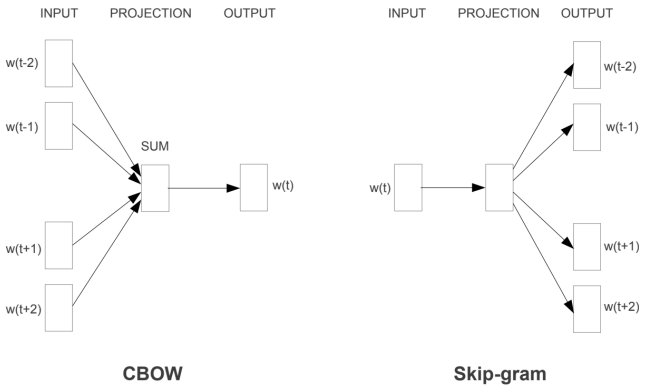


Figure 2.7: Word2Vec model architectures

The CBoW architecture is built to consider the surrounding words (context) around a target word (center) and is trained to try to predict it, creating a model capable of interpreting a full sentence and detecting a possible missing piece. Using as an example a classic sentence in NLP: *"the quick brown fox jumps over the lazy dog"*, this can create pairs of (context window, target word) where if considered a context window of size 2, it creates the following pairs: *[([quick, fox], brown), ([the, brown], quick), ([the, dog], lazy), etc.]*. Thus the model tries to predict the target word based on the context window words by finding the possible relationships between them based on the occurrence.

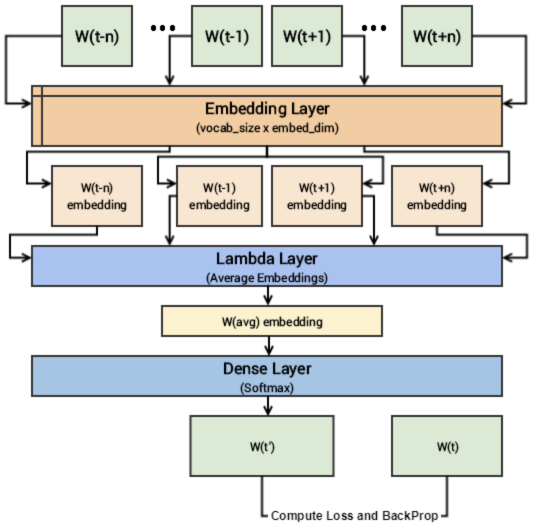


Figure 2.8: CBoW Neural Network layers

An overview of the deep learning layers [22] used for this method can be found in Figure 2.8, where is it possible to see that input context words (with a window size

of 2) are passed through a unique embedding layer with the desired representation size (initialized with random weights) and creating a dense word embedding for each of the context words. These embeddings are then merged together by summing or averaging them, obtaining a last dense layer which will be used to predict the target word.

The Skip-gram model strategy can be considered as the reverse process of the CBoW model. This architecture will be trained trying to predict the context words that surround a known target word. Following the example stated before with a context window of size 2, what this model would do is try to predict *[quick, fox]* given the word *fox*.

In order to train this model it is needed to identify each context word pair and label them as positive examples, while also creating negative examples by picking random target words and associating them with the context word pairs to predict.

As summarized in [22], the architecture of this NN can be seen in Figure 2.9 where the input to the neural network will be a one hot encoded version of the target word and the output is the one hot encoded version of the context word. Hence the size of input and output layers is V (vocabulary count). If it is a positive sample the word has contextual meaning, is a context word and our label Y=1, else if it is a negative sample, the word has no contextual meaning, is just a random word and our label Y=0.

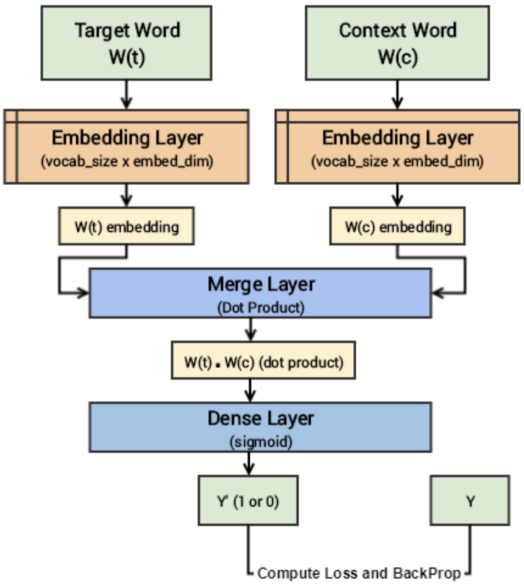


Figure 2.9: Skip-gram Neural Network layers

Both the target word and the context word pair will be passed through embedding layers with the desired representation size (initialized with random weights), and then will be merged performing a dot product of both matrices. This layer will be then used to predict the value of the label through a sigmoid activation.

An extension of W2V called FastText was proposed by Facebook [23] in 2016, where using the Skip-gram model as a base they improved its performance by representing each word as a bag of character n-grams and using that for training.

2.3.1.2 GloVe

As stated in [24] Global Vectors (GloVe) is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

This model was motivated by the idea that context window-based methods (such as W2V) do not take advantage of the knowledge that can be extracted by learning global corpus statistics (e.g. not learning repetition and large-scale patterns). Instead of being considered a predictive model such as W2V, GloVe is a count based matrix factorization model, where the initial step is to create a full word-context co-occurrence matrix consisting of (word, context) pairs like the one in Figure 2.10:

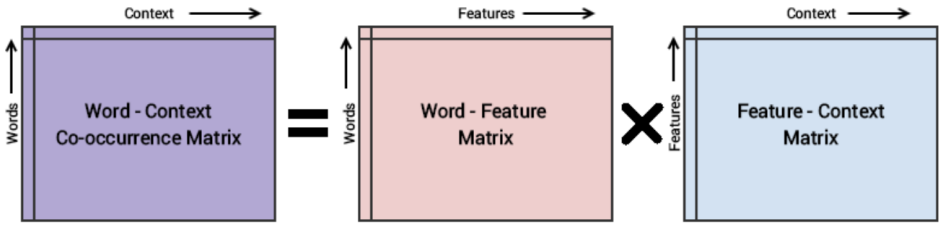


Figure 2.10: GloVe matrix construction

Considering the Word-Context (WC) matrix as the objective (created by making one pass on the corpora and representing how often a word appears in the context of another word), matrices Word-Feature (WF) and Feature-Context (FC) are multiplied aiming to reconstruct WC. For this, WF and FC are typically initialized with random weights and then iteratively multiplied using SGD as their optimization method to reduce reconstruction error. Finally, the Word-Feature matrix (WF) produces the word embeddings for each word where F can be preset to the desired lenght of the embedding.

2.3.2 Embedding layer

In order to apply the word embeddings techniques for ANN training, it is possible to consider some feature engineering techniques that would reshape the embeddings into an useful shape.

Basic forms of this are merging together all the embeddings for each word by summing or averaging them, creating a unique vector of fixed length no matter the quantity of words the text had. Other option can be to measure the difference between pieces of text (e.g. Q&A), considering that we can represent each piece of text as a vector, any distance formula (i.e. euclidean) could obtain valid information of how similar the texts are.

A more complex, but better, form of using embeddings inside ANN is the creation of an embedding layer inside the NN. This layer will have the capacity to map words from the text into an embedding matrix that contain the respective embedded word vector, and access the latent information from the word embedding during training.

This embedding layer can be used in different ways, one as a way to create word embeddings by training a model and adjusting the vector weights (leading to a model

that can be reused later), other as a way to have another set coefficients to optimize and learn in a bigger model, and lastly as a method of transfer learning by learning previously saved word embedding model as new features.

In order to create the embedding layer, the words will have to be mapped to an unique integer ID that will represent the index in the embedding matrix. After that, the matrix weights will have to be initialized (by random initialization or by loading previously saved weights). The layer will be created as 2-dimensional (index vector, embedding matrix) and will handle the input data and coefficient adjustments internally while training.

2.4 Neural Language Model

As described in [25]: statistical language modeling is the task of assigning a probability to sentences in a language. In addition to assigning a probability to each sequence of words, the language models also assigns a probability for the likelihood of a given word (or a sequence of words) to follow a sequence of words. The application of language modeling is part of the Natural Language Generation (NLG) subsection of NLP.

An NLM is a language model based on ANN, that takes advantage of their natural capacity of learning distributed representation of information while handling big quantity of features with a reduced impact of the curse of dimensionality. In their basic form, NLM are train for the sole task of learning to predict the next word after a text sequence.

In a way, NLM try to learn representations of text based on the use of language (grammar, semantic, etc.) and when trained with massive corpora they manage to find hidden aspects of how words relate between each other based on context and probability. Most of the NLP advances in the last years have come from the creation and sharing of bigger, more powerful and optimized pretrained NLM models.

2.4.1 Bidirectional Language Model

In [26] a variation of LSTM was proposed in a way that surpassed most of the NLP state of the art benchmarks at that time. This model took advantage of bidirectional RNN where during training the feed forward step uses the memory of the words before the target token and the backpropagation step uses the memory of words that come after the target token.

This architecture type, called BiLSTM, is the heart of the proposed model Embeddings from Language Model (ELMo). The motivation for ELMo is that word embeddings should incorporate both word-level characteristics as well as contextual semantics.

Figure 2.11 displays how the BiLSTM passes around the information from the embedding to later output the resulting encoded vector with a deeper level of information. An example to understand the improvement in information, could be comparing the word embedding for the word *"play"* created with W2V or ELMo. In the case of W2V the resulting embedding would have the same value no matter the contextual use of the word *"play"* as the word is considered unique and no disambiguation is possible, while in the BiLSTM layer of ELMo the word *"play"* could create different embeddings

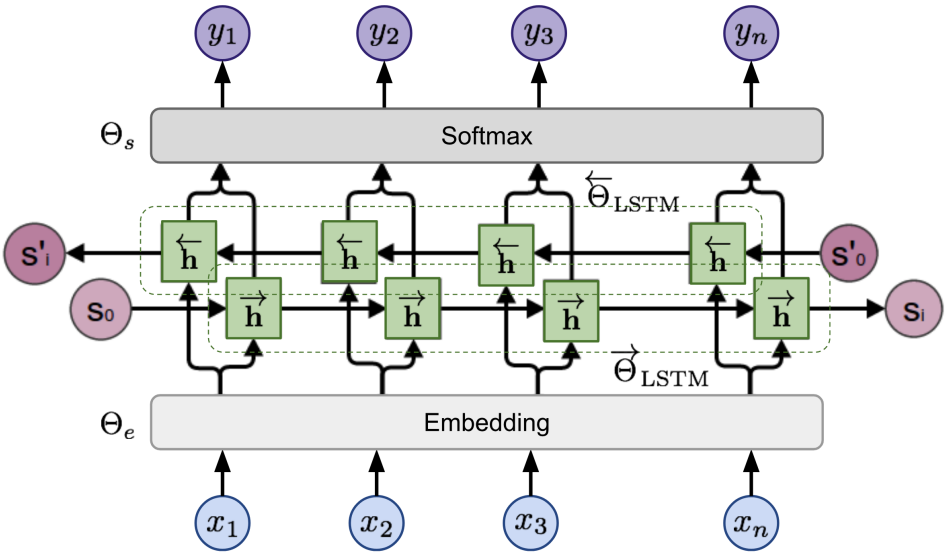


Figure 2.11: BiLSTM architecture

if feed with two examples like: *"I went to see the Lion King play at the theater."* and *"I went to see the lion play at the zoo."*.

ELMo will learn to represent the word in its context and, because of this, will learn parts of the semantic behind it. In order to do this, ELMo will represent the word as a weighted combination of the embedding and the hidden layers obtained from the BiLSTM, creating a deep representation.

It was empirically shown in [26] that the higher level states (right to left direction on the sequence) of the BiLSTM capture context (thus having better accuracy for word sense disambiguation) and that the lower level (left to right direction) captures syntax (performing better on POS tagging tasks).

2.4.2 Attention

When considering the translation of long sentences, a human method to do this would be to understand the whole sentence and pay attention to the key words, for then constructing the translated sentence around the relevant words adjusting its order to the grammar rules of the desired language.

The concept of Attention in NLP has provided of a new technique on how to improve RNN limitations and also further improve performance of Neural Machine Translation (NMT). Even with the improvements of LSTM, working with long sequences of text on RNN still can prove challenging as the memory cells eventually do not learn the importance of the information they were holding in the general picture.

An RNN can attend over the output of another RNN. At every time step, it focuses on different positions in the other RNN and by focusing everywhere (to different extents) it makes the calculation differentiable, thus allowing optimization inside the network. An example of this architecture can be seen in Figure 2.12

As described in [27]: the attention distribution is usually generated with content-based attention. The attending RNN generates a query (e.g. a subset of the target

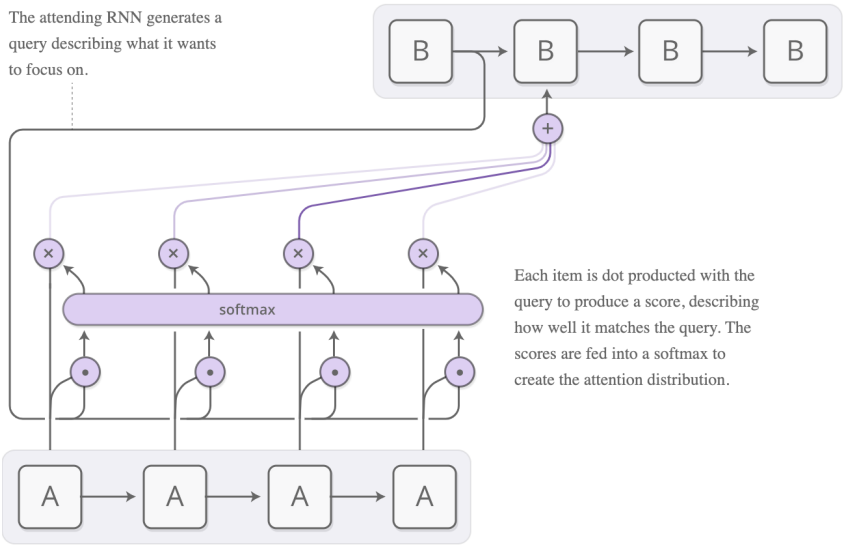


Figure 2.12: Attention process

information) describing what it wants to focus on (e.g. similarity between the query key and the comparison key). Each item is dot-product (or another similarity function) with the query to produce a score, describing how well it matches the query. The scores are then fed into a Softmax activation to create the attention distribution.

2.4.3 Encoder-Decoder with Attention

On the field of NMT, the addition of Attention has greatly improved the quality of translations [28]. Taking a popular model of NMT, the Encoder-Decoder architecture (i.e. Seq2Seq), Attention provided a way to give weights and semantic relevance to key words to focus in the given context and helped to obtain a better translation of the meaning of the sentence.

This can be achieved thanks to the fact that Attention can be calculated as an intermediate step in between time steps of the RNN of Seq2Seq and then used to impact the output.

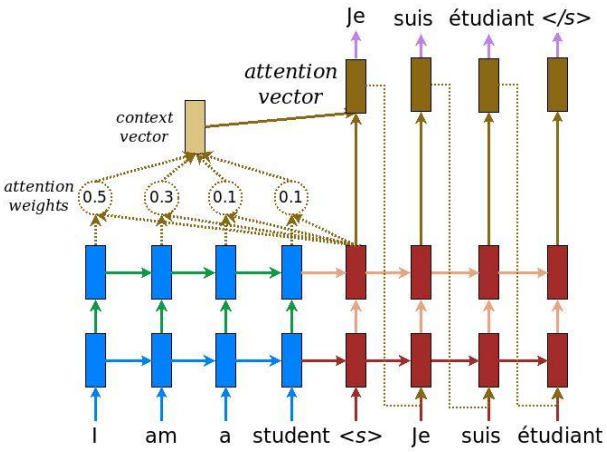


Figure 2.13: Seq2Seq with Attention

In Figure 2.13 it is possible to see an example of a sentence going through the architecture of Seq2Seq, where the Encoder tries to translate the input sequence into a comprised numerical form, and then through the Decoder, which is feed the Encoder continuous representation of the translation and has its weights initialized with the last state of the Encoder, the conversion from numerical representation into words is done. For this enhanced version of Seq2Seq the addition of Attention will assign an Attention weight to each word which will then be used by the decoder to predict the next word in the sentence. After this, the loss of the translation can be calculated by comparing the reconstruction error, allowing the full Encoder-Decoder model start optimizing its parameters to learn how to translate.

Attention requires the creation of three vectors: Query, Key and Value, which will be trained and updated during the training process. In terms of Encoder-Decoder, these three vectors can be described as: the Query being the hidden state of the Decoder, the Key as the hidden state of the Encoder, and the Value as the normalized weight, representing how much attention a Key gets.

2.4.4 Self-Attention

Another evolution on Attention was proposed in [29], Self-Attention, which proposes that is an attention mechanism relating different positions of a single sequence (i.e. word) in order to compute a representation of the sequence (i.e. the full sentence it belongs to). This means that in comparison with regular Attention, the query will be modified as it will only consider information from its same sentence.

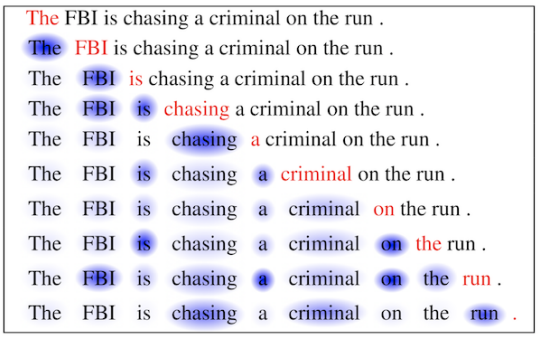


Figure 2.14: Self-Attention scores step by step

An example of how Self-Attention scores the attention of each word while traversing a sentence word by word be seen in Figure 2.14. This was demonstrated by [30] and it displays how for a current word (in red) the rest of the past words behave in terms of attention (size of the blue shade indicates the activation level).

2.4.4.1 Transformers

The Transformer architecture published in [31] is the last key component of the current state of the art in the field of NLP. This type of architecture was designed to improve the computer efficiency of the logic involved in the model Seq2Seq, which due to its sequential processing can not be parallelized. Initially this was attempted by employing convolutional layers (which are naturally parallelizable) into the RNN and pooling through the sentence, but it computational cost was still to expensive to improve speed.

The proposal was to focus the model solely on Attention, leaving behind some of the architecture limitations of RNN, while also achieving better results thanks to the newfound capacity of stacking more layers without exponentially raising the computational costs. One of the reasons of this performance improvement is obtained by having a deeper architecture focused on Attention which gives the ability to better process long range dependencies inside sentences.

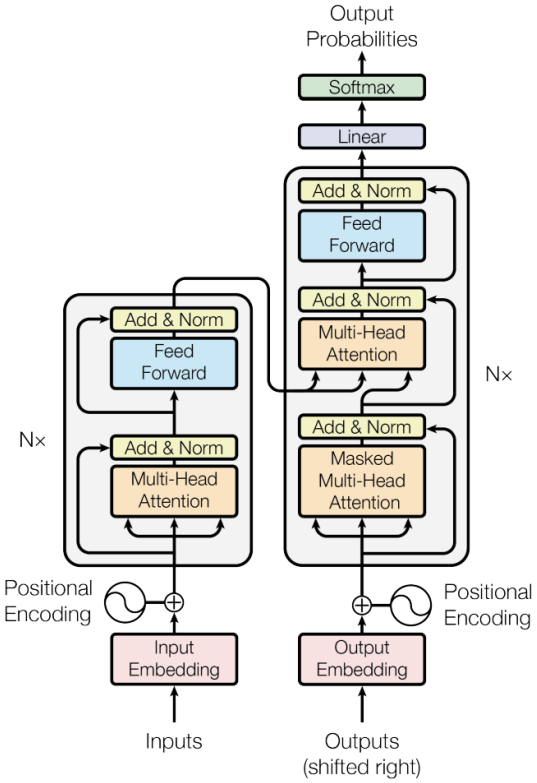


Figure 2.15: Transformer architecture

As seen in Figure 2.15, the initial input will be the embedding of the sequence with additional information on absolute (or relative) position of each token in the sequence (positional encoding). The fact of not passing through the sentence in a sequential manner (i.e. word by word in each step like an RNN would do), forces the model to have a fixed input size and map their positions to not lose context.

Once the embedding has been properly encoded it will be the input of the encoder layer. Transformer has several encoders stacked (quantity defined by an hyperparameter, where the proposed model had 6 stacks) the first one being feed the encoded embeddings and then propagating its output to the next one, with the last encoder feeding the attention vectors (Key and Value) into all of the decoder stacks at the same time for its initialization. Both the encoder and decoder layer stacks have the same components in their architecture.

The novel addition of the Transformer comes from the Multi-Head Attention (Figure 2.16). Heads will be multiple Attention layers running in parallel through chunks of the sequence (8 in the paper), and will serve as dimension reduction representing a linear projection of the Key, Query and Value, which will later concatenated and linearly transformed. They will calculate self-attention for words in different parts of

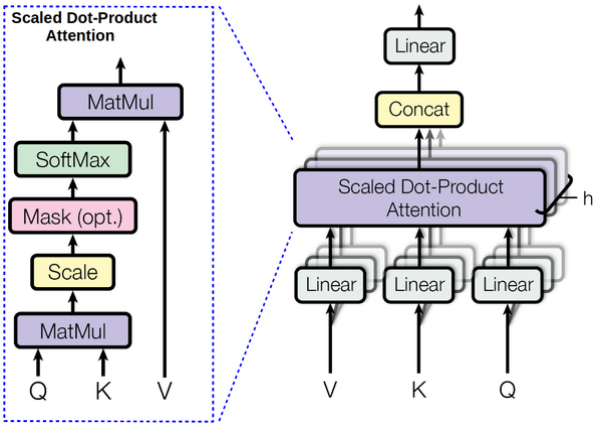


Figure 2.16: Transformer's Multi-head Attention

the input sequence and then scalar dot-product the attention between each other. As described in [31]: "Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions."

The fully connected Feed Forward Network (FFN) will receive the aggregated output of the Multi-Head Attention layer and the addition of a residual connection that offers information about the position related information from previous tokens. This combination of information will be normalized and then feed into the FFN, which is composed by two linear activation functions and 1 ReLU for each position.

After the FFN, any new possible residual information will be merged again, and then everything will be propagated into the decoder step. In Figure 2.17 it is possible to see an example of how a sentence composed by two tokens would create a set of two FFN that would process each word with Attention information inside the full context for the Encoder.

The process inside the decoder will proceed in a similar manner with the input coming from the encoder's output, except that all Multi-Head Attentions will become a Masked Multi-Head Attention layer to enforce next token prediction. This modified layer will have masked future positions of the sentence (forcing them to $-\infty$) before the Softmax activation inside of the Attention head, thus only feeding the decoder with past tokens. Another addition will come from a new Attention layer after the Multi-Head Attention coming from the merger of Attentions calculated through the encoder and the decoder.

While training the decoder it will also be possible to use a technique called Teacher Forcing which will help to achieve faster convergence (Encoder-Decoder models also can take advantage of this). What this technique will do is sequentially judge every proposed token from the encoder and if incorrect, replace it with the ground truth value to stop the next position from carrying the error. In order to achieve this, the input of the decoder will be have to be shifted right by 1 token (the START token), thus allowing the Encoder to give the first prediction.

The decoder stack will produce a vector of floats, which the final Linear layer will project into a vector of the same size of the known output vocabulary (logits vector) where each position corresponds to the score of a unique word. These scores will then be transformed into probabilities by the Softmax layer and the chosen next word will

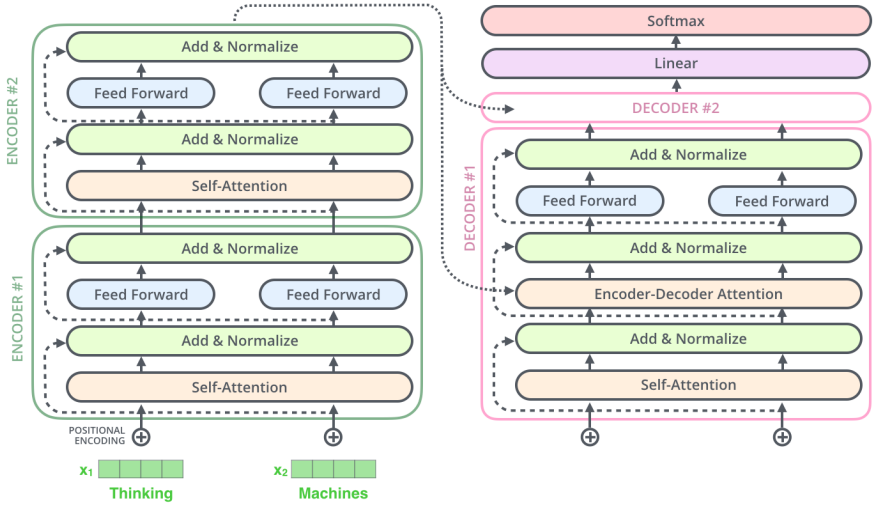


Figure 2.17: Transformer's Encoder-Decoder architecture

be the one with the biggest probability.

2.5 Natural Language Understanding

This sub-field of NLP focuses exclusively on tasks related to machine reading comprehension. This type of problem is considered Artificial Intelligence (AI) complete or hard, meaning that its solution requires the creation of real computational intelligence due to its complexity being so close to cognitive and human reasoning.

Some of the main tasks of NLU involve machine translation, text automated reasoning, content analysis and categorization, and question answering, among others. In a general sense, NLU focus on anything related to creating knowledge from the information found on text inside the language context (e.g. associated word emotion on sentiment analysis).

While there are several NLU specific algorithms (e.g Latent Dirichlet Allocation for topic modelling), a big component of this field comes from techniques also associated with other fields of NLP. Theories and algorithms developed for the fields of IR or NLG have proven to bring general improvements on state of the art results when adapted into NLU tasks. To date it is possible to see adapted models based on basic BiLSTM, ELMo, Attention and Transformers performing at the top of performance of the GLUE benchmark [2].

2.5.1 Universal Sentence Encoder

Universal Sentence Encoder (USE) [32] is a pretrained model developed by Google and made publicly available on 2018. USE aims to help encoding sentences into high dimensional vectors that already contain previous information beneficial for transfer learning fine-tuning for specific tasks (e.g. text classification).

USE can be considered as an expansion of word embeddings capable of working at a sentence contextual level and creating a result with a fixed length thanks to an

encoding process that employs only the encoder segment of the Transformer network architecture.

In order to achieve a pretrained model capable of performing for general NLU tasks, USE is trained with three different problems: SkipThought (which works on learning fixed length numerical representations of heterogeneous sentences), a conversational input-response task (learn extraction of information from parsed conversational data) and classification tasks (trained on supervised data).

As mentioned in the publication, USE's capability of creating vectors that contain embedded information from the words in their context, in addition to the transferred knowledge obtained from the extensive pretraining, while also adding the information from the word embeddings (i.e. W2V) has proven to be a successful mechanism for feature engineering while training classification algorithms.

2.5.2 Bidirectional Encoder Representations from Transformers

Developed by Google and published through their own Deep Learning library TensorFlow, Bidirectional Encoder Representations from Transformers (BERT) [5], was defined of being: "designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks".

Is this deep bidirectional representation that makes BERT models address one of the limitations of architectures like ELMo, where bidirectionality was performed in a shallow manner (caused by the fact that its training considered the input left to right and right to left separately, only joining it at the end). The extensive use of Transformers with a fully interconnected network and the use of a novel training method are the main reasons behind the success of BERT on several of the popular NLP benchmark tasks.

The training methodology and the input data used for it make any pretrained BERT model capable of easily be fine-tuned for almost any downstream NLP task without the necessity of using big training datasets for fine-tuning. This is possible thanks to the general knowledge obtained from deeper and intimate understandings of how the language works inside of very large text corpus used for training: the entire Wikipedia (with 2500 million words) and BookCorpus (800 million words).

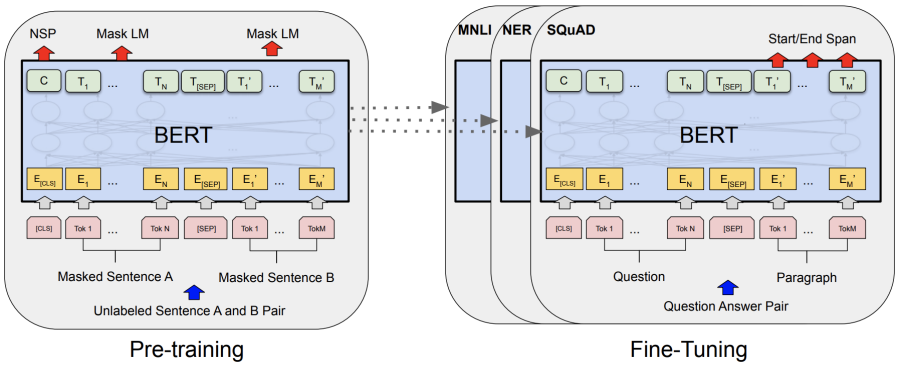


Figure 2.18: BERT architecture task based

The architecture of BERT is based on the original implementation of the Transformer, only using the Encoder sub-network and fully connecting each stacked layer between each other, creating a multi-layer bidirectional Transformer encoder with, initially, two predefined model sizes:

- BERT-Base: 12 layers (Transformer blocks) with 12 Attention-Heads and 768 hidden nodes, which produces a total of 110 million trainable parameters, trained through 4 days on 4 Cloud TPUs.
- BERT-Large: 24 layers (Transformer blocks) with 16 Attention-Heads and 1024 hidden nodes, which produces a total of 340 million trainable parameters, trained through 4 days on 16 Cloud TPUs.

In all cases, the maximum sequence length of the Embedding vector feed as input is of 512 tokens.

2.5.2.1 Pre-training BERT

As BERT uses bidirectional Attention (meaning that in the case of the decoder step the future tokens are not masked as in a regular Transformer) it can cause that the words are able to "see themselves" through other connected layers while working on different tokens. In order to avoid this information leakage, two novel semi supervised training methods were created for BERT.

The first is Masked Language Model (Masked LM), which masks a percentage of the tokens (15%) and forces the model to predict the masked words (target labels). An input example could be:

- Input: *the man went to the [MASK1] . he bought a [MASK2] of milk.*
- Labels: *[MASK1] = store; [MASK2] = gallon*

The masked words are selected from a 80% of the tokens, while another 10% is replaced by a random word and the last 10% is kept intact. Adding this noise while training forces the encoder to learn representative vectors of each token in a generalized way, helping the robustness of the model for transfer learning when fine-tuning.

The second method, Next Sentence Prediction (NSP), will be given two sentence and train the model to predict if the next sentence is likely to be a continuation of the previous one or not. The training negative samples are created randomly and their ratio with real sentence pairs is 1:1. Examples of this use case are:

- Input: *[CLS] the man went to the [MASK] [SEP] he [MASK] a gallon of milk [SEP]*
- Label: *IsNext*
- Input: *[CLS] the man [MASK] to the store [SEP] [MASK] are flight ##less birds [SEP]*
- Label: *NotNext*

This method helps the model learn the relationship between pieces of text by the addition of the auxiliary tokens (CLS and SEP), which provide a way to detect the boundaries of the two sentences and let the model understand better the information given by both sentence representation in relation with the target.

2.5.2.2 BERT input representation

At the moment of publication all versions of BERT for the English language had a vocabulary of little more than 30000 tokens with their respective embedded representation pretrained on the published model.

Tokens in BERT are created from parsing text with the WordPiece tokenizer, which will split words up to a minimal recognized form and keep the rest marked as a suffix morpheme. For example, the word *"flightless"* will be tokenized into *flight* and *##less* (the prefix ## means the token was obtained after a split). The intuition behind this is that BERT will learn the representation of the basic form of the words and also understand the modifications that different language rules can have on it.

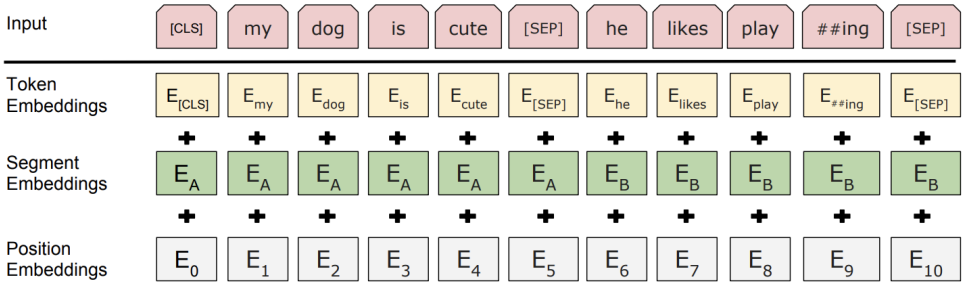


Figure 2.19: BERT inputs

In Figure 2.19 is possible to see the input format that BERT is expecting to receive, based on its architecture and the requirements associated with it. The preprocess involved on this will be to first tokenize the text, then trim it up to the maximum length desired (padding with 0 for shorter sequences) and lastly create the three input embeddings:

- Token Embeddings: IDs pointing to the associated WordPiece token vocabulary, which will retrieve the embeddings learned for the specific token.
- Segment Embeddings: mechanism to differentiate sentence pairs (e.g. Q&A), which leads to learning a unique embedding for the first and the second sentences to help the model distinguish between them.
- Position Embeddings: since being based on Transformers, BERT requires a way to inform the sequence of the tokens, using positional embeddings to express the position of words in a sentence and learn from their order.

2.5.2.3 Fine-tuning BERT

For the goal of fine-tuning BERT for specific tasks the model offers several tweaked architectures, each one for a specific problem (e.g. classification, multiple choice selection and Q&A). This can be done by just adding a specific task head on top of the BERT architecture and using its output as prediction. In general terms, BERT provides three predefined outputs [33]: pooled head, sequence head and hidden states.

The pooled head is a built-in classifier (used while pre-training NSP), that will use the last layer hidden state of the first token of the sequence (i.e. [CLS] token) and make it go through a Linear layer and a TanH activation, resulting in a prediction within a range of [-1, 1].

The sequence head provides the raw output of the last layer's hidden states for the entire sequence, which can then be averaged or pooled to use it as input for a custom classifier head (i.e. MLP) that will optimize its weights based on the BERT representation while training with the new data.

The hidden states comprise the selection of the layers of Transformer blocks inside the BERT architecture (e.g. 12 for BERT-Base), plus the output of the initial embedding used to feed the Transformer. As the heads are stacked in sequential manner, each head can offer different knowledge from the input, for instance, the initial layers will have a less refined representation of the text, focusing better on the tokens that have the most impact. Same as the sequence head, these outputs can be later aggregated and used for a custom classifier.

With the desired architecture, it will also be possible to decide if the weights inside the BERT layers should be optimized by training or not. This process will give the liberty to freeze any layer, just using the knowledge obtained from the pre-training for the creation of the representation, and will highly reduce the cost of computation for the fine-tuning task.

Several efforts have been done to further improve the performance obtained with a Fine-Tuned BERT for classification. In [7], they encourage a series of recommendations to try while training for a specific task that involve: long text handling (instead of trimming the sequence, taking pieces from the beginning and the end), hyperparameters calibration, variable learning rate for each layer, usage of the different BERT outputs and specific heads, further training BERT for a specific domain problem. Also, another useful proposal can be found in [8], where they improve classification performance through enriched meta-data (i.e. information related to the text in its domain) added as a new layer on the custom classification head on top of BERT.

2.5.2.4 Related models

The publication of BERT triggered an incredible amount of research and the creation of new models (with 16 official models published [34] at the moment of writing). These models take concepts used in BERT and either customize the inside architecture, use a different method of preprocessing the text, use different pre-training techniques or simply take some of the ideas and mix them with other models.

Considering that one of the main drawbacks from BERT is its massive size which requires powerful computer resources (mainly memory limitations), neural network distillation became a perfect fit for it. AlBERT [35] and DistilBERT [36] are the two main versions of smaller BERT that uses distillation for its compression, while also tweaking some of its functionality (e.g. cross layer parameter sharing) to regain some of the loss of performance due to the compression.

Other successful models that have achieved state of the art results are XLNet [37], which counts with the addition of the improved transformer Transformer-XL [38], overcomes deficiencies of the Masked LM method and adds other optimizations; and also RoBERTa [39] which was Facebook's effort of porting BERT into their Deep Learning framework (PyTorch) and had an additional and more extensive pre-training process by using more data.

Chapter 3

Development and Method

This chapter will define some of the rules and procedures taken into account for the experimentation and results analysis. As part of a Kaggle competition, specific conditions have to be consider that might limit overall performance or measurement capacities. Also, being the case of a complex NLU problem, special focus is required on the development environment with its DL framework and respective resource handling.

3.1 Setup

Based on the delivery conditions set by Kaggle, the development of this research was done on online platforms with their associated computing resources and libraries. The language of choice was Python through Notebooks and in most cases GPU computation was activated.

3.1.1 Development environment

Kaggle [40] and Google [41] offer free online developing platforms only with usage limitations for long run-time executions. At the moment of writing, on Kaggle's side there is a weekly quota of 30 hours of GPU power and executions have to be shorter than 9 hours, while in Google Colaboratory executions are only limited to 12 continuous hours with a wait time of around 12 hours once that limit is reached (does not have a weekly quota).

Final models have to be delivered through Kaggle platform and, to avoid competition cheating, the notebook will not have internet connection, forcing to use the files and libraries already installed on the platform or pre-installing them after approval by the competition moderators.

Kaggle's platform offers Jupyter Notebook interface, while Google's has a customized version of it. In terms of hardware they have similar specifications as both use Nvidia Tesla P100 16GB as their primary GPU (falling back to Nvidia Tesla K80 12GB when limited resources are available) and an Intel Xeon dual core CPU. The main difference comes with RAM memory (Kaggle has 13GB and Google 26GB) and hard drive (Kaggle 5GB, while Google 68GB with transparent connection to Google Drive).

3.1.2 Python dependencies

To maintain cross-functionality between platforms, the same libraries and versions were used in Kaggle Notebook and Google Colaboratory. The Python version selected was 3.6 and the main libraries and versions used are listed as follow:

- *numpy (1.18.2)*: essential library for matrix and algebraic operations.
- *pandas (0.25.3)*: provides useful methods to handle data (extract, transform and load).
- *scikit-learn (0.21.3)*: principal Data Science library with multiple tools for feature engineering, pipelines, modeling and more.
- *scipy (1.4.1)*: scientific module with statistic, optimization and algebra functions.
- *nltk (3.2.5*: suite of libraries for NLP. Additional downloaded packages: stopwords, punkt, averaged_perceptron_tagger, wordnet.
- *gensim (3.6.0)*: library containing NLP tools. Popular for their implementation of Word2Vec and FastText.
- *transformers (2.4.1)*: main library for most current NLP/NLU models. Offers access to any Transformer based model and more.
- *tensorflow-gpu (2.1.0-rc0)*: Google's Deep Learning framework focused on differentiable programming of tensors.
- *tensorflow-hub (0.7.0)*: extension to TensorFlow to access online shared models.
- *keras (2.2.5)*: wrapper library that simplifies the usage of several Neural Network packages (e.g. TensorFlow).
- *torch (1.4.0)*: machine learning library developed by Facebook under the PyTorch project.
- *apex (0.1)*: Nvidia's extension tool for PyTorch that allows mixed precision calculation.

3.1.3 Neural Network libraries

The two main Deep Learning frameworks at the moment are TensorFlow and PyTorch and while they both focus on tensor computation (i.e. algebraic object that describes relationships between algebraic objects) and use Python as their primary language interface, they differ in terms of development and configuration.

TensorFlow offers a high level of customization, requiring to develop several components that manipulate and coordinate the process of training an ANN. One of them is the static computational graph which requires to be declared prior running the model, often adding complexity when working with variable length inputs such as text on RNNs.

The Deep Learning wrapper Keras can be used in order to have a more user friendly interface to work with TensorFlow. This library will provide methods that simplify several of the standard, yet complex, steps of training a model with TensorFlow, allowing powerful models to be trained in few lines of code focused only on the ANN

architecture. A drawback of using Keras comes from this simplicity, which often reduces the capacity to customize models and test new techniques.

PyTorch can be considered to be in between the development complexity of Keras and TensorFlow, but it retains most of the liberty of customization and also offers similar or better performance in NLP. A main difference compared to TensorFlow/Keras comes from the capacity to define and manipulate the computational graph on-the-go (i.e. dynamic graph), meaning that development can be done in a sequential form instead of declarative programming.

As part of working with Transfer Learning, using the right version of the model for the DL framework is needed. In the case of BERT, the specific saved model (BERT-Base uncased) had to be downloaded for Keras [42] and PyTorch [43], and also the related vocabulary [44] for the WordPiece tokenizer.

Due to the overall size of BERT, with a saved model size of 400MB and 4GB when initialized in memory, strict memory management has to be put in play to avoid any issues and low computation performance. Here is where mixed precision (i.e. performing calculation with 16 bit floating point instead of 32 when possible) can help to achieve faster GPU computing process and also use less memory.

This research will initially use a mix of Keras and PyTorch for the baseline models, but will later focus on PyTorch with Apex for the final fine-tuning thanks to the speed up, better memory management and code customization.

3.2 Experimentation method

As part of a short-term competition there were several particular considerations at the moment of experimentation. A strategy to perform well and obtain high scores is to initially test several baseline methods and select the best performing that still has room for improvement and start tweaking and calibrating it while testing new ideas.

This approach will be the one detailed in this work, and it will involve testing 4 different models (evolving in complexity), to later select the best one and focus on fine-tuning it through hyper-parameter calibration, new features or improved text representation, architecture optimization and any model related methods than apply.

The initial exploration will be performed inside of Kaggle's platform, while the final model will be extensively fine-tuned and trained on Google Colaboratory and then migrated into Kaggle only for inference.

3.2.1 Constraints

The competition was defined as "offline notebook only", meaning that the delivery will be a notebook unable to connect to any online resources and capable of running from start to end in a time limit of 2 hours when using GPU computation or 9 hours when using CPU.

For baseline models exploration this limit will be taken into account when performing both training and inference on the test set, while for the process of fine-tuning, the models will trained for longer periods (up to 12 hours) and then will have to be able to perform only inference in a time lower than 2 hours.

Another restriction comes from Kaggle's effort to discourage any type of cheating through test set probing. In order to do this, only a 13% of the test dataset with 476 unlabeled observations is made public, while the remaining private 87% is kept for the final scoring of the competition.

The method to learn how well the models perform will be based initially on their validation metrics (which will have the most level of detail) and later, through a final submission into Kaggle only the averaged Spearman correlation will be obtained for both test sets: the public and private (only obtained after the competition finished).

3.2.2 Validation techniques

Repeatability is a key factor while performing this type of study. In order to compare models or find improvements through customization, ensuring that each execution is deterministic is of extreme importance. In order to assert that, the following function (with the seed 21937) was used in all models:

```
def set_seeds(seed):
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    tensorflow.random.set_seed(seed)
    numpy.random.seed(seed)
    random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
```

During training, the input data used will be split into train and validation sets, leaving the hold-out set used by Kaggle to score the model as the test set. At the time of determining which data will be used to train (i.e. data used by the model to learn) and which for validation (i.e. data used to compare model calibration performance), several methods can be used: naive training (train and validate with all the observations), train/test split (select a majority percentage for training and leave the rest for validation) and cross-validation (create several folds with different train/test splits and train a model for each pair).

To obtain the best performance during training, cross-validation will be used and for the baseline comparison a GroupKFold is performed. The grouping used for the folds ensures that observations with the same question, but different answer, remain in the same split (thus, stopping any information leakage between train and validation).

3.2.3 Performance metrics

The end goal of this research is to obtain the best performing model based on the competition's metric: Spearman ρ correlation coefficient. This metric is a non-parametric measure of rank correlation (statistical dependence between the rankings of two variables) [45].

Figure 3.1 displays how a ρ correlation of 1 is obtained when all data points of a given x value have a greater y value. This assess how correlated the rank values between two variables is, as it is tried to be described through a monotonic function (linear or not). In contrast, the Pearson correlation does not achieve a value of 1, since it only considers linear relationships.

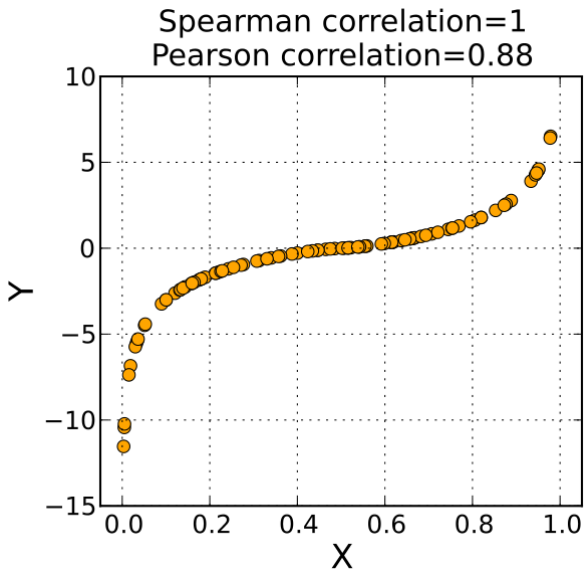


Figure 3.1: Spearman and Pearson correlations

Although the case at hand involves an ordering problem with a ranking metric, as a way to add more details while training, it is possible to also obtain information from the BCE loss as it would inform the regression/classification error.

Taking into consideration the constraints and validation techniques in place, the performance metrics to be used with each model will be:

- Training loss for each epoch per fold
- Validation averaged Spearman ρ for each epoch per fold
- Validation Spearman ρ of each label for the best epoch per fold
- Averaged Spearman ρ for public and private test sets

3.2.4 Baseline models

The selected models to explore for this problem will follow the evolution of techniques detailed in Chapter 2. Initially a simple MLP with features extracted from the text through NLP techniques will be tested (TF-IDF), then an LSTM with embedded word representation (GloVe), after that an MLP with transformer word representation (USE) and lastly, the BERT model.

The methodology used to train these models will be to keep their architecture as vanilla as possible, while also using the default hyper-parameters proposed when published, avoiding to explore any additional normalization and optimizer techniques and using the same loss function (BCE).

All baseline models will be trained with the same 5 folds obtained from the GroupKFold split (with 4863 train and 1216 validation samples). Models will run for a number of epochs that fits all the folds into the time limit, or until there is no improvement for two consecutive epochs.

In the case of the two first models (MLP and LSTM), only the best epoch for each fold will be used for inference, and their predictions will be averaged at the moment of

submission for the test set. Models that rely on Transfer Learning (USE and BERT), will average predictions from all epochs, as they do not train for several epochs and do not tend to have a cold start problem, offering good results from their initial iteration.

Chapter 4

Baseline experimentation and Results

The following chapter presents the initial studies and approach taken with the given problem and the available data. Several models are tested with their metrics being recollected for further analysis in the results discussion.

While the final goal to solve in this research is to rank each Q%A for each label ordered as precisely as possible, due to ordering being non-differentiable while training an ANN there is a need to use a proxy for learning how to rank.

Based on the fact that the labels are continuous values from 0 to 1, a possible solution could be to design a regression problem that will try to predict as close as possible each label score. Another option, could also be considering this problem as a binary classification (0 or 1) for each label and then using the predicted probability of being the positive case as the final value.

For the baseline study, it was decided to present a classification problem, thus using BCE loss function and a Sigmoid activation to fit values in between the desired range.

4.1 MLP model

This initial model [46] will cover the basic concepts that involve NLP study over text. It will focus on heavy feature engineering, trying to create new variables that capture different levels of hidden information present in the corpus. Relying mainly on content description, linguistic rules and IR concepts, the obtained dataset will be feed in a simple MLP network.

4.1.1 Features

A pipeline was designed to create a set of new features that would extract information from the three text features (*question_title, question_body, answer*) and two categorical features (*host, category*). Also, all text features were pre-processed creating another set of three cleaned text features (removal of stopwords, HTML code, non-standard symbols and lemmatization).

In summary, the dataset is composed by the one hot encoded version of the categorical features, plus the following concepts applied to all 6 resulting text features:

- TF-IDF with SVD to compress to 150 features
- Count of: characters, words, unique words, sentences, punctuation, question words, numbers
- Ratio of: POS Tags, stopwords, uppercase letters, word match between text pairs

This produces a dataset with around 2800 features which, additionally, will be normalized and standardized as a whitening process to improve MLP training convergence.

4.1.2 Model

The designed ANN architecture can be described as follow:

1. Input layer with whitened features
2. Weight normalized hidden layer with feature compression into 128 nodes
3. ReLU activation
4. Weight normalized hidden layer with size of 128 nodes
5. ReLU activation
6. Weight normalized output layer with size of 30 labels
7. Sigmoid activation

The used loss function was *BCE* and the optimizer selected was *Adam*. The configuration used to train the model had the following hyper-parameters:

- *Learning rate*: 1e-3
- *Batch size*: 1
- *Maximum number of epochs*: 500
- *Patience*: 5 epochs for early stopping
- *Computation method*: CPU (time limit 9 hours)

4.1.3 Training metrics

With a total runtime of 22 minutes, the obtained metrics for this model show that convergence into the proximity of a minimum generally takes around 80 epochs with steady optimization, and, as seen in Figure 4.1, both metrics reach an asymptotic point where small improvements are achieved through slowly iterating over more epochs.

The validation Spearman ρ tends to flatten at the end of each fold's model, showing how the lack of improvement triggered the early stop of the training. Additionally, considering that the training's BCE loss also flattens at that point suggests that the model had reach its maximum potential.

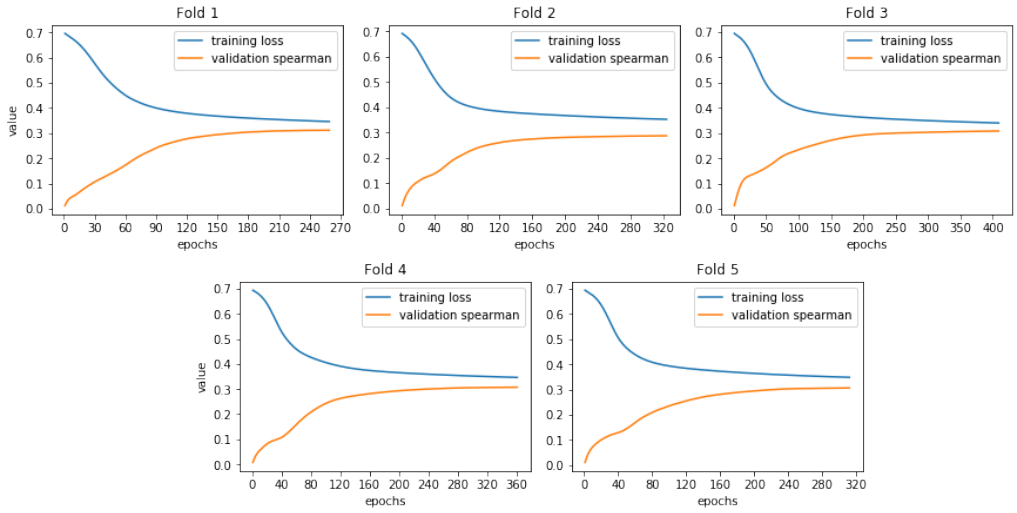


Figure 4.1: MLP model metrics

The fact that the model struggles to overfit, which would be the case if the training loss would still further decrease through iterations while the validation metric worsen, indicates that the features engineered and the ANN architecture lack the capacity to create a function capable to find a solution to this complex dataset.

	Fold 1	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**
question_asker_intent_understanding	0.3287	0.2993	0.2574	0.1592	0.2369
question_body_critical	0.5538	0.5869	0.5703	0.5260	0.5748
question_conversational	0.3224	0.3233	0.3586	0.3662	0.2980
question_expect_short_answer	0.2243	0.1929	0.1372	0.2022	0.2156
question_fact_seeking	0.2046	0.2726	0.2162	0.2495	0.2058
question_has_commonly_accepted_answer	0.3461	0.3153	0.3832	0.3044	0.3167
question_interestingness_others	0.2940	0.2843	0.3209	0.3121	0.3350
question_interestingness_self	0.3947	0.3970	0.3953	0.4309	0.4480
question_multi_intent	0.3436	0.3200	0.3736	0.3662	0.3936
question_not_really_a_question	0.0287	-0.0147	-0.0184	-0.0252	0.0471
question_opinion_seeking	0.2825	0.3333	0.2879	0.2844	0.2889
question_type_choice	0.5273	0.5385	0.5237	0.5435	0.5346
question_type_compare	0.2739	0.2785	0.2362	0.2669	0.2231
question_type_consequence	0.0538	0.0924	0.1016	0.1299	0.0795
question_type_definition	0.3126	0.2630	0.3071	0.3233	0.3259
question_type_entity	0.3856	0.2169	0.3400	0.2975	0.3192
question_type_instructions	0.7037	0.6192	0.7084	0.7046	0.6723
question_type_procedure	0.2661	0.2431	0.2803	0.2996	0.2263
question_type_reason_explanation	0.4764	0.4536	0.4990	0.5535	0.5179
question_type_spelling	0.0358	0.0132	0.0432	0.0545	0.0667
question_well_written	0.4227	0.3744	0.4826	0.3986	0.4924
answer_helpful	0.1994	0.1058	0.1500	0.1334	0.1422
answer_level_of_information	0.3257	0.3603	0.3616	0.3448	0.3687
answer_plausible	0.0947	0.0389	0.0881	0.0451	0.0582
answer_relevance	0.1060	0.0811	0.1037	0.0831	0.1105
answer_satisfaction	0.2755	0.1682	0.1983	0.2167	0.1757
answer_type_instructions	0.6894	0.6291	0.6920	0.6865	0.6604
answer_type_procedure	0.2313	0.1655	0.1931	0.2552	0.2258
answer_type_reason_explanation	0.5567	0.5248	0.5630	0.6188	0.5589
answer_well_written	0.0805	0.1273	0.0792	0.0891	0.0775

Table 4.1: MLP model labels' Spearman ρ

When analyzing in Table 4.1 the capacity of the model to order each label by fold, it is possible to see how some of the unbalanced labels have extreme poor performance (e.g. *question_not_really_a_question, question_type_spelling* and some balanced ones (like answer_well_written) also have bad results. Based on the scores for *question_asker_intent_understanding* between folds, it might be possible that fold 4 got a bad distribution of observations that did not allow the model to learn a pattern as well as in the other folds. On the other hand, Q&A related to instructions (i.e. *question_type_instructions, answer_type_instructions*) clearly have key features and a balanced label that proved easy for the model to interpret.

4.2 LSTM model

This model [47], based on a BiLSTM architecture with Attention, introduces the use of NLM techniques and provides a first glance of how powerful sequential representation of embedded text can be. The key component of this model will be its ANN architecture, which will learn from the text representation created from GloVe embeddings.

4.2.1 Features

In order to create the dataset to feed this model, each of the three text features will have to be encoded into fixed size embeddings. To do so, first text will be cleaned (removal of symbols and expansion of word contractions), then word tokenized and the resulting arrays trimmed (maximum length of 50 for *title_question* and 500 for *title_body, answer*).

The trimmed arrays will be padded on the end, if needed, and the embeddings will be created through the values obtained from GloVe Common Crawl 840B Tokens (with a 2.2 million vocabulary and a vector representation of length 300).

The final form of the dataset will be composed by the three GloVe embedded text features and the addition of one hot encoded version of the categorical features (*host, category*).

4.2.2 Model

The designed ANN architecture can be described as follow:

1. Embedding layer for each of the three text features encoded by GloVe
2. Dropout layer for each of the three embeddings
3. Bidirectional LSTM layer with 256 nodes for each three layers from 2
4. Bidirectional LSTM layer with 128 nodes for each three LSTM layers from 3
5. Attention layer for each of the 3 LSTM layers from 4
6. Concatenation of the Averaged and Max pooled layers from 4 and the Attention layer from 5
7. ReLU activation for each layer from 6
8. Concatenation of categorical features with all layers from 7

9. Linear combination and Sigmoid activation

The used loss function was *BCE* and the optimizer selected was *Adam*. The configuration used to train the model had the following hyper-parameters:

- *Learning rate*: 1e-3
- *Batch size*: 32
- *Number of epochs*: 15
- *Patience*: 2 epochs for early stopping
- *Dropout*: 0.4
- *Computation method*: GPU (time limit 2 hours)

4.2.3 Training metrics

In the case of this model's performance, the training was limited to less epochs than expected as it would start overfitting on longer training periods, and in cases like folds 4 and 5 training was cut even earlier due to lack of improvement as seen in Figure 4.2. In total the runtime of this model was of: 66 minutes.

It is possible to observe how the embeddings used in this model and the learning process of a BiLSTM with Attention already give good results almost from the first epoch, in contrast to the MLP model's metrics where the initial epochs had scores close to 0, having an initial performance without any cold startup issues.

Whereas the MLP model had a stable general behaviour across folds, this model shows different training processes on each fold, and also does not manage to converge smoothly. An even lower learning rate might help stabilize and train this model for longer epochs, but as seen in all folds, it is clear that there is a limit of how much improvement on the final score can be.

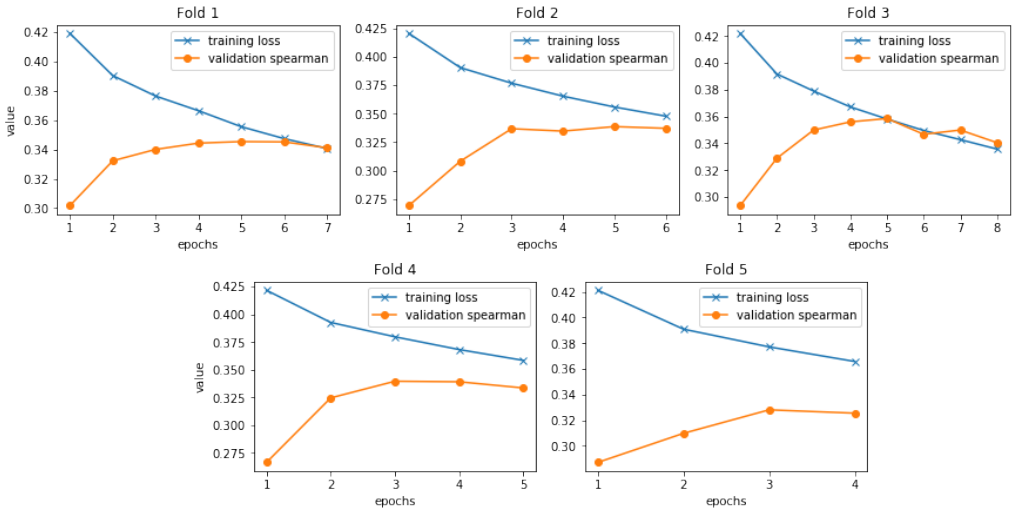


Figure 4.2: LSTM model metrics

In Table 4.2 it is possible to see a similar pattern as in the MLP model of where the model struggled to order better. Again, unbalanced labels *question_not_really_a_*

question, *question_type_spelling* had the worst results across all folds, and the top ones where labels related to instructions. An improvement can be seen in *question_asker_intent_understanding* where the scores achieved were closer to the general average of the model and the variation between folds was lower.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
question_asker_intent_understanding	0.3705	0.3348	0.3762	0.3000	0.3256
question_body_critical	0.6123	0.6741	0.6238	0.6053	0.5842
question_conversational	0.3603	0.3774	0.4264	0.3826	0.3240
question_expect_short_answer	0.2062	0.2473	0.2354	0.2293	0.2424
question_fact_seeking	0.2724	0.2974	0.2896	0.2591	0.2299
question_has_commonly_accepted_answer	0.3875	0.3621	0.4377	0.3478	0.3454
question_interestingness_others	0.3414	0.3282	0.3426	0.3710	0.3117
question_interestingness_self	0.4738	0.4681	0.5005	0.4777	0.4756
question_multi_intent	0.5125	0.5239	0.5221	0.5196	0.5092
question_not_really_a_question	0.1011	0.0818	0.0946	0.0476	0.0828
question_opinion_seeking	0.3465	0.3696	0.4003	0.3494	0.2867
question_type_choice	0.6237	0.5942	0.6392	0.6644	0.6000
question_type_compare	0.3093	0.2914	0.3191	0.2865	0.2577
question_type_consequence	0.1051	0.0962	0.1054	0.1101	0.1329
question_type_definition	0.3482	0.2656	0.3201	0.3275	0.3394
question_type_entity	0.3919	0.3025	0.3123	0.3031	0.3548
question_type_instructions	0.7233	0.6608	0.7180	0.6960	0.6756
question_type_procedure	0.3077	0.2195	0.2601	0.2730	0.2304
question_type_reason_explanation	0.4783	0.4317	0.4599	0.5156	0.5045
question_type_spelling	0.0466	0.0687	0.0602	0.0577	0.0859
question_well_written	0.4570	0.4883	0.5220	0.4375	0.4946
answer_helpful	0.2421	0.1916	0.2106	0.1584	0.1790
answer_level_of_information	0.3541	0.3791	0.4171	0.3884	0.3817
answer_plausible	0.1050	0.1524	0.1235	0.1129	0.0770
answer_relevance	0.0977	0.1613	0.1283	0.1164	0.1239
answer_satisfaction	0.2104	0.2717	0.2425	0.2177	0.2207
answer_type_instructions	0.6961	0.6627	0.7092	0.7015	0.6850
answer_type_procedure	0.1915	0.2003	0.2030	0.1767	0.2147
answer_type_reason_explanation	0.5592	0.5248	0.5520	0.5729	0.5845
answer_well_written	0.1278	0.1379	0.1249	0.1089	0.1346

Table 4.2: LSTM model labels' Spearman ρ

4.3 USE model

The USE model [48] offers an introduction to the usage of Transformers and how to leverage their information representation from text. The process involving this ANN is straight forward, with just an MLP capable of learning from the encoded text input. In order to boost performance, the initial layer of the ANN will use the same parameters (i.e. sharing weights and biases), so general knowledge from all the text features can be learned.

4.3.1 Features

The pre-process involves the creation of the three main text features transformed through the embedding function of USE Large. The benefit of using the encoding of USE is that all text information will be comprised into a fixed length vector, thanks to the capability of the Transformer's encoder.

The dataset will be composed by vectors of length 512 for *question_title*, *question_*

body, answer. Additionally, one hot encoded version of the categorical features (*host, category*) are added.

4.3.2 Model

The designed ANN architecture can be described as follow:

1. Layer with shared weights and betas for each of the three text features encoded by USE
2. Dropout layer for each of the three layers from 1
3. Layer without shared parameters for each of the three layers from 2
4. Dropout layer for each of the three layers from 3
5. Dropout for additional layers with categorical features
6. Concatenation of categorical features with all layers from 4
7. Linear combination and Sigmoid activation

The used loss function was *BCE* and the optimizer selected was *Adam*. The configuration used to train the model had the following hyper-parameters:

- *Learning rate*: 5e-4
- *Batch size*: 64
- *Number of epochs*: 15
- *Patience*: 2 epochs for early stopping
- *Dropout*: 0.2
- *Computation method*: GPU (time limit 2 hours)

4.3.3 Training metrics

While analyzing the performance of this model through Figure 4.3 it is possible to see the potential of information representation through Transformers. With a total runtime of 13 minutes, the fact that the model has a steady decrease in its training loss while the validation metric remains flat, indicates that overfitting is possible and that, in a way, the dataset and architecture used have the potential to find a specific function that could explain at least the training set.

Although searching for model generalization with more normalization techniques could be possible, it is likely it will not boost any further the performance of the model as it seems to peak constantly around ρ: 0.37 in all folds and the loss starts showing the previously mentioned overfitting trend.

As in the LSTM model, folds 4 and 5 again prove to be the more difficult ones for the model to learn. Using Table 4.3 it can be explained that this happens by having a general slight lower score in good performing labels and the existence performance gaps between folds in labels as: *question_not_really_a_question, question_fact_seeking, question_type_compare*. The rest of the folds have a general better scoring average compared to previous models, with at least five labels with constant scores above 0.6 and other five labels with values between 0.4 and 0.6.

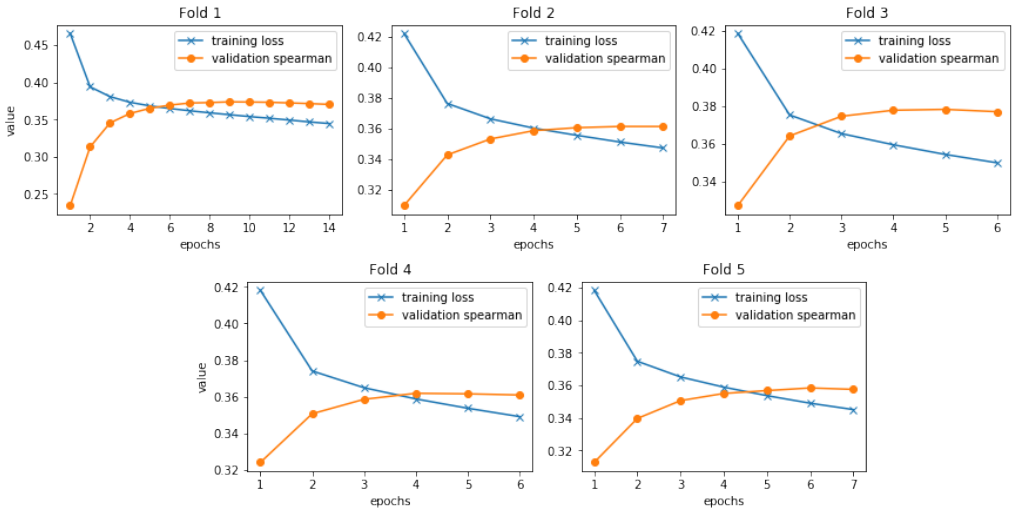


Figure 4.3: USE model metrics

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
question_asker_intent_understanding	0.3853	0.3631	0.3586	0.2982	0.3525
question_body_critical	0.6177	0.6368	0.6136	0.6465	0.5953
question_conversational	0.3789	0.3782	0.4395	0.4149	0.3657
question_expect_short_answer	0.2509	0.1971	0.2351	0.1748	0.2030
question_fact_seeking	0.3036	0.2631	0.3249	0.2577	0.2331
question_has_commonly_accepted_answer	0.4228	0.3520	0.4207	0.3270	0.3693
question_interestingness_others	0.3545	0.3427	0.3722	0.3479	0.3557
question_interestingness_self	0.4767	0.4489	0.5108	0.4834	0.4780
question_multi_intent	0.3477	0.3395	0.3317	0.3676	0.3428
question_not_really_a_question	0.1356	0.0841	0.0600	0.0195	0.0377
question_opinion_seeking	0.3797	0.3649	0.4196	0.3396	0.3649
question_type_choice	0.6189	0.6300	0.6095	0.6143	0.5560
question_type_compare	0.3790	0.3203	0.3580	0.3082	0.2998
question_type_consequence	0.1578	0.1236	0.1711	0.1692	0.1735
question_type_definition	0.3640	0.2796	0.3379	0.3580	0.3725
question_type_entity	0.4715	0.3734	0.3728	0.3645	0.4475
question_type_instructions	0.7667	0.7357	0.7442	0.7379	0.7213
question_type_procedure	0.3024	0.2921	0.2893	0.2886	0.2929
question_type_reason_explanation	0.5863	0.5749	0.5774	0.6260	0.5965
question_type_spelling	0.0482	0.0701	0.0638	0.0671	0.0831
question_well_written	0.4531	0.4475	0.4992	0.4298	0.4641
answer_helpful	0.2105	0.1887	0.1684	0.1748	0.1534
answer_level_of_information	0.3673	0.3721	0.3647	0.3744	0.3615
answer_plausible	0.1132	0.1356	0.1116	0.1333	0.1320
answer_relevance	0.1162	0.1691	0.1586	0.1132	0.1186
answer_satisfaction	0.2999	0.3066	0.2613	0.2849	0.2679
answer_type_instructions	0.7422	0.6939	0.7404	0.7212	0.7064
answer_type_procedure	0.2560	0.2382	0.2336	0.2347	0.2442
answer_type_reason_explanation	0.6139	0.6273	0.6435	0.6578	0.6194
answer_well_written	0.1445	0.1249	0.1347	0.1229	0.1328

Table 4.3: USE model labels' Spearman ρ

4.4 BERT Classifier model

The use of BERT with pooled output [49] intends to show the potential an ease of use of this state of the art model. A notorious impact while using Transfer Learning mod-

els comes from how minimalist network architectures still achieve great performance, while also leaving room for improvement.

4.4.1 Features

Considering BERT's input requirements, all text features will have to be join together into the same input. In order to do so, a simple method will be to trim each text and concatenate it with the [SEP] token, so the model understands its Q&A form.

For this case, *question_title* will be limited to 30 tokens, while *question_body, answer* will have a maximum length of 239. In order to achieve this, each text feature will be tokenized by the WordPiece tokenizer and then trimmed and padded.

As a way to try to fit as much tokens as possible, in case one of the text features has less tokens than the limit, its total length will be reduced to that quantity (without padding) and the surplus will be given to the next text feature total length.

The final form will then be: [CLS] token, *question_title* tokens, [SEP] token, *question_body* tokens, [SEP] token, *answer* tokens, and [SEP] token; creating a fixed length of 512. Also, segment and position arrays will be created based on the token embeddings generated.

4.4.2 Model

The designed ANN architecture can be described as follow:

1. Input layers with token embeddings (WordPiece tokenized text), segment embeddings and position embeddings
2. BERT layer with pooled output (classification)
3. Sigmoid activation

The used loss function was *BCE* and the optimizer selected was *Adam*. The configuration used to train the model had the following hyper-parameters:

- *Learning rate*: 3e-5
- *Batch size*: 8
- *Number of epochs*: 3
- *Computation method*: GPU (time limit 2 hours)

4.4.3 Training metrics

One of the things that stand out while analyzing the metrics of this model in Figure 4.4 is how the validation score increases at a fast pace. Thanks to the architecture that leverages transfer learning, BERT manages to offer from its first epoch almost the same results as previous models, and clearly benefits from the pre-trained weights by adjusting them in each epoch to obtain even better results.

The main disadvantage of the model comes from its runtime, with one epoch every 6 minutes and a total of 94 minutes, makes it impossible to add another epoch for all folds and still fit under the time limit. Still, as seen in Table 4.4 the performance for

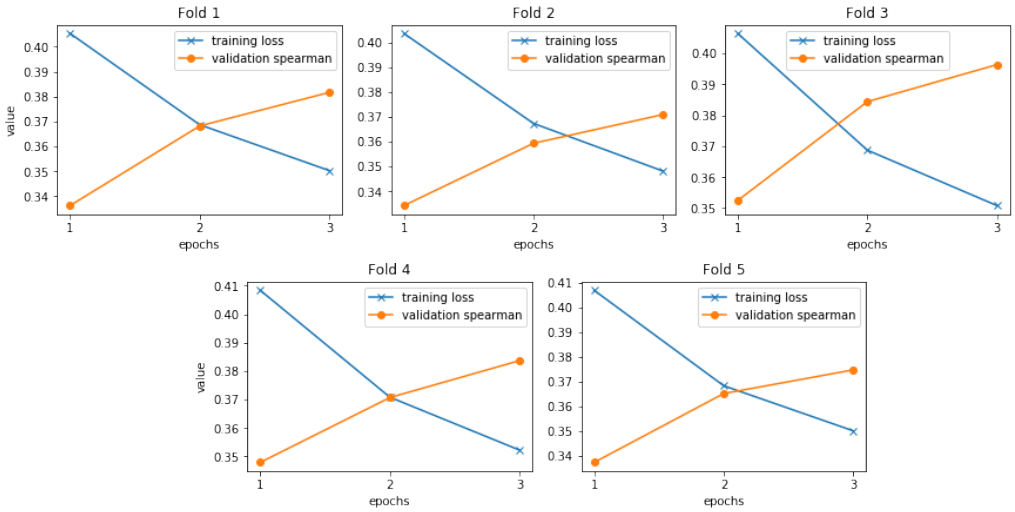


Figure 4.4: BERT Classifier model metrics

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
question_asker_intent_understanding	0.4042	0.3801	0.3653	0.3163	0.3477
question_body_critical	0.6192	0.6265	0.5836	0.5956	0.5646
question_conversational	0.3782	0.3854	0.4539	0.4282	0.3535
question_expect_short_answer	0.3065	0.2876	0.3169	0.3020	0.3051
question_fact_seeking	0.3342	0.3669	0.3781	0.3399	0.2785
question_has_commonly_accepted_answer	0.4192	0.4134	0.4768	0.3975	0.4102
question_interestingness_others	0.3116	0.3055	0.3394	0.3440	0.3436
question_interestingness_self	0.4638	0.4271	0.5227	0.4677	0.4993
question_multi_intent	0.5212	0.4995	0.5261	0.5919	0.5355
question_not_really_a_question	0.0586	0.0757	0.0480	0.0237	0.0405
question_opinion_seeking	0.4123	0.4652	0.5381	0.4411	0.4164
question_type_choice	0.7195	0.7278	0.7252	0.7289	0.7048
question_type_compare	0.3718	0.3531	0.3594	0.3158	0.3028
question_type_consequence	0.1333	0.1396	0.1850	0.1945	0.1688
question_type_definition	0.3641	0.2873	0.3612	0.3750	0.3730
question_type_entity	0.5082	0.4020	0.4206	0.3760	0.4708
question_type_instructions	0.7891	0.7757	0.7903	0.7751	0.7555
question_type_procedure	0.3647	0.3337	0.3358	0.3838	0.3229
question_type_reason_explanation	0.6295	0.6330	0.6685	0.7000	0.6560
question_type_spelling	0.0142	-0.0160	0.0490	0.0676	0.0854
question_well_written	0.4714	0.4894	0.5239	0.4478	0.5250
answer_helpful	0.2105	0.1731	0.1726	0.1525	0.1433
answer_level_of_information	0.3023	0.3549	0.3396	0.3397	0.3313
answer_plausible	0.0633	0.0624	0.0958	0.1123	0.1208
answer_relevance	0.1429	0.1308	0.1358	0.1440	0.1377
answer_satisfaction	0.2590	0.3148	0.2704	0.2649	0.1872
answer_type_instructions	0.7636	0.7281	0.7731	0.7586	0.7528
answer_type_procedure	0.3222	0.2412	0.2853	0.2948	0.2532
answer_type_reason_explanation	0.6407	0.6406	0.6901	0.6920	0.6599
answer_well_written	0.1502	0.1219	0.1593	0.1383	0.1948

Table 4.4: BERT Classifier model labels' Spearman's ρ

each label follows a similar behaviour as in other models, but with a general boost on scores that provide an improvement on the average Spearman ρ.

As a side note, testing the model outside of the time limit shows that improvements can be achieved for at least two more epochs with the same pace, boosting scores for

an average of 0.015 ρ.

4.5 BERT Sequence model

Knowing before hand the limitations of customization that the BERT classifier output has, this model [50] will be a way to validate if the performance of using the sequence output has room for improvement. The power on this model will come from directly using the encoded information created from the BERT representation and manually optimizing a classifier with it. Due to the inside parameters of BERT not being frozen, each iteration will not only train the classifier, but will also optimize the inside weights of BERT, creating a more fine-tuned model for this specific problem.

4.5.1 Features

For this experiment, the features used will be the same as the ones detailed in Subsection 4.4.1.

4.5.2 Model

The designed ANN architecture can be described as follow:

1. Input layers with token embeddings (WordPiece tokenized text), segment embeddings and position embeddings
2. BERT layer with sequence output
3. Averaged pooled layer from 2
4. Dropout layer from 3
5. Linear combination and Sigmoid activation

The used loss function was *BCE* and the optimizer selected was *Adam*. The configuration used to train the model had the following hyper-parameters:

- *Learning rate*: 3e-5
- *Batch size*: 8
- *Number of epochs*: 3
- *Dropout*: 0.2
- *Computation method*: GPU (time limit 2 hours)

4.5.3 Training metrics

As seen in Figure 4.5 and Table 4.5, the behaviour of this model is similar to BERT Classifier, but an important difference can be seen on how even the initial epoch has already a boost of at least 0.015 ρ compared to the metrics observed in Figure 4.4.

This model has the same time limitation issue as BERT Classifier, also with a runtime length of 94 minutes. But, the fact that the inside weights can be adjusted better in regards of the dataset labels, allows for a better learning process of the classification layer added at the end of the sequence output, finally reaching the 0.4 ρ mark.

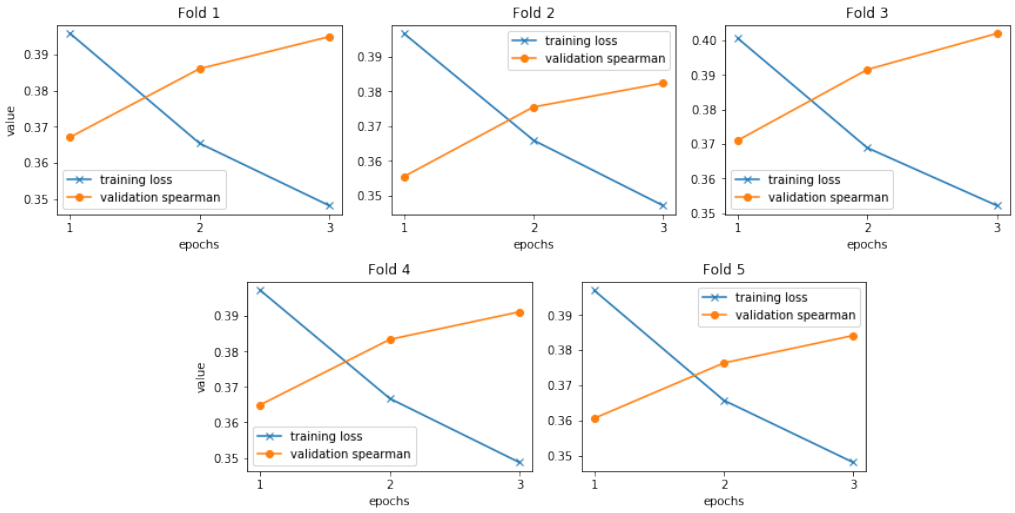


Figure 4.5: BERT Sequence model metrics

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
question_asker_intent_understanding	0.3963	0.3646	0.3764	0.3219	0.3657
question_body_critical	0.6255	0.6164	0.5578	0.5736	0.5720
question_conversational	0.3861	0.3833	0.4426	0.4422	0.3655
question_expect_short_answer	0.3005	0.2917	0.3141	0.3144	0.3133
question_fact_seeking	0.3633	0.3561	0.3759	0.3298	0.2924
question_has_commonly_accepted_answer	0.4170	0.4193	0.4738	0.4078	0.4307
question_interestingness_others	0.3472	0.3213	0.3206	0.3489	0.3592
question_interestingness_self	0.4617	0.4251	0.5364	0.4947	0.5297
question_multi_intent	0.5426	0.5399	0.5436	0.5973	0.5536
question_not_really_a_question	0.0605	0.1093	0.0567	0.0652	0.0450
question_opinion_seeking	0.4325	0.4698	0.5302	0.4246	0.4173
question_type_choice	0.7172	0.7237	0.7245	0.7336	0.7108
question_type_compare	0.3572	0.3450	0.3738	0.3353	0.3175
question_type_consequence	0.1464	0.1592	0.1671	0.2049	0.1736
question_type_definition	0.3751	0.2968	0.3632	0.3740	0.3803
question_type_entity	0.5157	0.4051	0.4241	0.3907	0.4683
question_type_instructions	0.7890	0.7717	0.7856	0.7807	0.7516
question_type_procedure	0.3703	0.3390	0.3420	0.3788	0.3135
question_type_reason_explanation	0.6438	0.6394	0.6631	0.6876	0.6485
question_type_spelling	0.0325	0.0513	0.0383	0.0687	0.0807
question_well_written	0.4754	0.4950	0.5103	0.4546	0.4945
answer_helpful	0.2261	0.2253	0.2304	0.1954	0.1856
answer_level_of_information	0.3693	0.3878	0.3952	0.3803	0.4112
answer_plausible	0.1105	0.1049	0.1177	0.1068	0.1073
answer_relevance	0.1625	0.1483	0.1569	0.1561	0.1454
answer_satisfaction	0.3097	0.3224	0.3265	0.2878	0.2326
answer_type_instructions	0.7699	0.7291	0.7654	0.7575	0.7510
answer_type_procedure	0.3080	0.2670	0.3122	0.3109	0.2683
answer_type_reason_explanation	0.6572	0.6394	0.6721	0.6898	0.6617
answer_well_written	0.1786	0.1259	0.1633	0.1196	0.1771

Table 4.5: BERT Sequence model labels' Spearman's ρ

4.6 Results

The final scores obtained after submitting each model can be seen in Table 4.6. Additionally, the averaged folds validation score was added as guideline.

As expected, all models suffer from negative impact on their private test score. This is caused by having to train a model with a small dataset of 6079 observations, which will struggle to generalize and learn complex patterns that can relate the text and the expected order of the labels' values. Having a private test dataset of more than half the train dataset size (with 3185 Q&A) will penalize any type of unlearned hidden pattern during training.

Model	**Validation score**	**Public test score**	**Private test score**
MLP	0.30396	0.30329	0.28831
LSTM	0.34207	0.34179	0.31580
USE	0.35577	0.36046	0.32842
BERT Classifier	0.38144	0.37498	0.35013
BERT Sequence	0.39092	0.37802	0.35504

Table 4.6: Baseline models final scores

Not surprisingly validation and public test scores tend to be similar, being the reason that the small public test set of 476 observations does not have a high variation in its content, thus allowing the model to achieve similar performance as the one obtained during training. Worth noting that this can be achieved thanks to using cross validation for training, which helped getting a more stable result in low scales by using all the information available to train and reducing any extreme overfitting.

In sync with the evolution of NLP techniques, the performance of each model is improved by the next one. Models like MLP and LSTM display having reached its maximum potential, with final scores that are clearly surpassed by the performance of initial epochs of models involving Transformers architecture. Although USE gives a promising public test score, its drop on private score suggest a lack of capacity to improve as any further training will worsen the overfitting behaviour.

Both BERT models clearly show a gap in performance compared with the rest, with the use of transfer learning embedded into the model's architecture as the key component of the potential of this model. As mentioned at the moment of publication in [5], fine-tuned BERT achieves good results even with small training sets.

Taking advantage of the pre-trained weights and BERT's text representation makes the model break the Spearman's ρ 0.4 barrier while training and as seen in model BERT Sequence, the validation scores shows promise. The goal of the fine-tuning process will be to boost as much as possible this score and then add normalization techniques that could help lower the negative impact of the private test set.

Chapter 5

Model fine-tuning

This chapter describes the series of tests and tweaking performed in order to fine-tune and improve BERT's performance for this dataset. Based on the initial quick study it was detected that a basic form of BERT offered the best results and as a next step finding any quick wins that can boost performance even further will be the focus of the testing.

Due to the competition's context and the lack of access to constant powerful computation resources, the research method will have to be adjusted into smaller iterations and a decision making process sometimes based on theoretical knowledge, previous published empirical studies, or in last resort, logical reasoning.

The initial objective is to test fundamental architecture changes that could open up to new tweaking opportunities and provide the biggest boosts possible. After that, minor changes can be searched in reduced scenarios that allow to a fast decision making process.

Once the best possible configuration has been found, the model will be trained exhaustively for a longer period of time and used for the final submission.

5.1 Tweaking tests

The main ideas to investigate will try to cover aspects detected through the baseline study and the understanding of how BERT works. As a way to speed up the process, while also having a good detail of the model's behaviour, tests will be performed on a reduced selection of folds (using folds 1, 3 and 5 as during the baseline study proved to behave differently).

A change from the BERT model used in the baseline is a complete code migration into PyTorch in Google Collaboratory, which makes customization more flexible and also allows the use of Apex for mixed precision calculation, thus reducing memory allocation from 4GB per BERT model to 3GB and making forward and backward passes of the network less memory consuming. A drawback of this migration, is that the pre-trained BERT model will not behave in the same way as the model trained during the baseline study, hence needing the creation of a new baseline.

The fine-tuning research will be performed in a incremental manner, meaning that ideas will be tackled sequentially and the best configuration will be used for the next

test. Due to the length of the training period (which will be between 10 to 15 minutes per epoch), exploring all possible combinations of tweaks is restrictive, and in some cases conflicting or not promising combinations will be avoided, trying to only test the most impactful changes. Since the end goal is to find an optimal configuration for the creation of the best model possible capable of doing inference in less than 2 hours, the training will be allowed to be for longer periods of time to find the model's potential.

The first test will intend to tackle BERT's fixed length limitation, trying to add as much of the text possible for training through additional layers. Then, the focus will be to use BERT's information representation in the best possible way by exploring its outputs. Lastly, in order to help the model generalize better, and avoid as much as possible the score drop in the private test set, improving regularization will be explored.

With the main architecture of the model decided, several minor tweaks and hyper-parameters will be calibrated with short runs (i.e. comparison of performance of initial epoch), leaving a series of successful adjustments and discarded tests. As a last improvement, and a way to boost the scores only for competition purposes, post-processing techniques will be implemented on the final model submission trying to overcome some of the model's deficiencies.

5.1.1 Dataset

Due to the Q&A being split into three segments (*question_title, question_body* and *answer*), the arrangement of a concatenation that can shape the text into BERT format causes the loss of valuable information. To fit into the maximum allowed the baseline proposal used lengths of: 30/239/239, which considering the token lengths of each text field, as seen in Table 5.1, limited information to almost a third.

Percentile	question_title	question_body	answer
75%	13	204	212
90%	16	345	360
94%	18	448	470
95%	18	496	514
99%	23	1043	993
100%	47	3077	8184

Table 5.1: Text features tokens length statistics

To overcome this limitation, the idea of splitting the question and the answer into two separate models was tested. In order to do this, two BERT pre-trained models will be initialized together, creating a new model architecture with their outputs combined in a final set of layers.

A new dataset structure had to be created, splitting *question_body* and *answer* as second sentence in the input and using *question_title* as the initial sequence. It can be said that BERT is able to understand the relationship between the title and the two second parts partly because the title can serve as a short question, with both, the body and the answer, as related pieces of text (or answers).

5.1.1.1 Test

A new dataset split into pairs of [*question_title, question_body*] and [*question_title, answer*] is created with maximum lengths of 50/459 that covers, in the worst case scenario, almost 95% of the available tokens.

The designed ANN architecture to work with this new dataset format can be described as follow:

1. Input layers with token embeddings (WordPiece tokenized text), segment embeddings and position embeddings for the two text pairs
2. Two BERT layers with sequence output for each text pairs
3. Concatenation of outputs from 2
4. Averaged pooled layer from 3
5. Linear layer of size 1536 with ReLU activation
6. Dropout layer from 5
7. Linear layer of size 768 with ReLU activation
8. Linear combination and Sigmoid activation

The used loss function was *BCE* and the optimizer selected was *AdamW*. The configuration used to train the model had the following hyper-parameters:

- *Learning rate*: 3e-5
- *Batch size*: 6
- *Number of epochs*: 5
- *Dropout*: 0.2
- *Computation method*: GPU (total runtime: 180 minutes)

5.1.1.2 Results

Considering that the migrated code to PyTorch has a different behaviour, a new model was created following the new experimentation settings. The results of this new baseline for the selected epochs can be seen in Figure 5.1, being the most notable difference the lower initial point achieved on the first epoch.

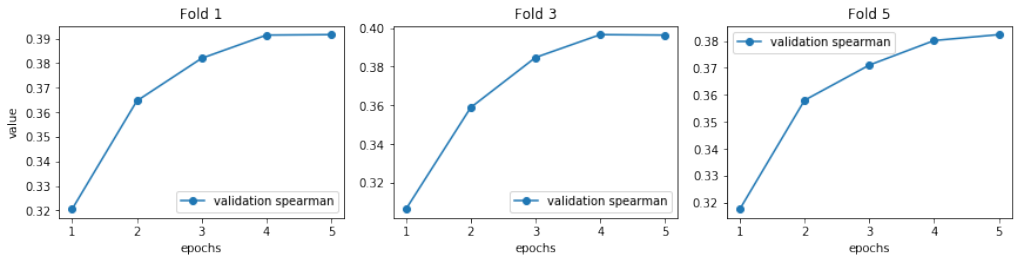


Figure 5.1: BERT PyTorch model metrics

The training metrics of the tested model with the dual architecture can be seen in Figure 5.2, and while the initial epochs between models have a similar trend, Dual

BERT is capable of obtaining higher scoring peaks thanks to the longer training period.

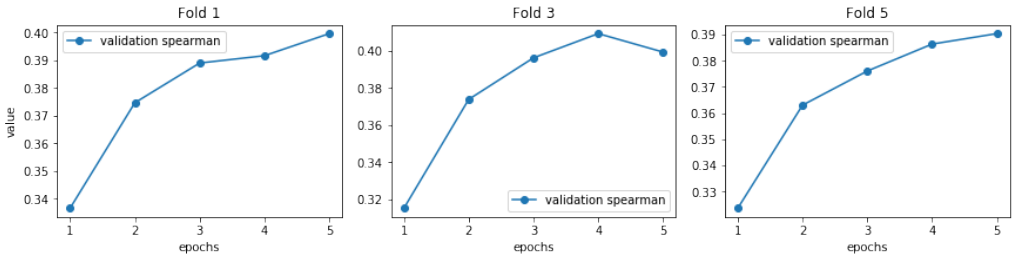


Figure 5.2: Dual BERT model metrics

The validation's averaged Spearman's ρ scores can be seen in Table 5.2 and the proposed model improves around 2.45% the validation performance, making it a valid improvement over the baseline in this scenario.

Model	Validation score
PyTorch BERT	0.39021
Dual BERT	0.39977

Table 5.2: Dual BERT validation score comparison

In order to finish calibrating this new model, other tests were performed with the following impact on the validation score in comparison with Dual BERT:

- Reducing *question_title* maximum length to 30: *-4.16%*
- Selecting fragments of both ends of the sentence instead of just the beginning (as suggested in [7]): *-4.44%*
- Addition of categorical features *host, category*: *+0.7%*

5.1.2 Architecture

As suggested by BERT creators on its publication, using the outputs from the inner BERT layers can offer a more variate encoded representation and lead to more powerful models.

Several tactics have been discussed over previously mentioned articles related to BERT: using the last layer, the second to last, last four layers or even all layers. If BERT's architecture is considered, it is possible to state that the last layer will give similar results as the sequence output and that the first layer might not produce good results as it is a layer mostly optimized for NSP. General consensus indicates that the performance of the selection of layers depends mostly on the problem to solve.

5.1.2.1 Test

The best configuration found was using the last four layers and performing a two step parameter compression through ReLU activated layers. This led to the creation of an ANN architecture that can be described as follow:

1. Input layers with token embeddings (WordPiece tokenized text), segment embeddings and position embeddings for the two text pairs
2. Two BERT layers using the last four heads as output for each text pairs
3. Concatenation of outputs from 2 (a total of eight layers)
4. Averaged pooled layer from 3
5. Linear layer of size 6144, compressing into size 1536 with ReLU activation
6. Concatenation of categorical features with layer from 5
7. Dropout layer from 6
8. Linear layer compressing into size 384 with ReLU activation
9. Linear combination and Sigmoid activation

The used loss function was *BCE* and the optimizer selected was *AdamW*. The configuration used to train the model had the following hyper-parameters:

- *Learning rate*: 3e-5
- *Batch size*: 5 (had to be decreased due to memory limitations)
- *Number of epochs*: 5
- *Dropout*: 0.2
- *Computation method*: GPU (total runtime: 225 minutes)

5.1.2.2 Results

The results of this model, as seen in Figure 5.3, give empirical validation to the statement previously mentioned as it can be seen that this new model has a clear boost on its initial epochs by being capable of optimizing faster, thanks from a more variate information representation.

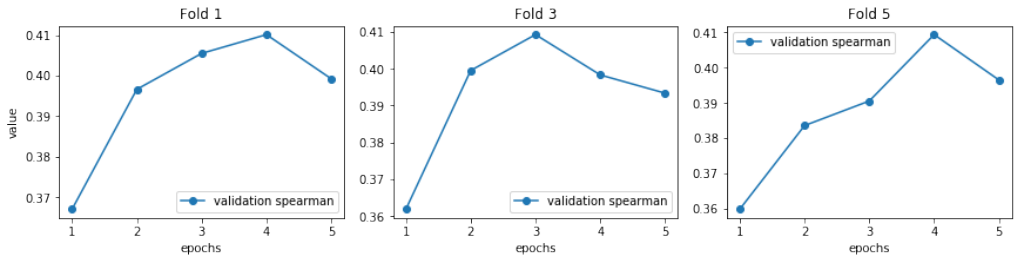


Figure 5.3: Multi Head BERT model metrics

A new finding is that based on this faster convergence, the model manages to achieve its peak before the last epoch and then proceeds to overfit and decrease the generalization performance, something almost not seen on the previous BERT models.

The validation's averaged Spearman's ρ scores can be seen in Table 5.3 and the new model improves 2.39% the performance, breaching the 0.4 ρ barrier for the first time.

Several other architecture tests were made, mostly achieving worse results:

Model	Validation score
Dual BERT	0.39977
Multi Head BERT	0.40959

Table 5.3: Multi Head BERT validation score comparison

- Using only second to last layer: *-2.53%*
- Using all 12 layers: *-3.17%*
- Altering the order of heads concatenation and layer pooling gave similar results
- Calibration of different layer sizes for compression or expansion led to worse results (around -0.6%)
- Additional layers on the MLP part gave similar results but increased model runtime and final size
- Concatenating results of max and average pooling instead of just using average pooling: -1.4%
- Freezing BERT layers speeds up training time 3X, but decreases performance by: -44.5%

5.1.3 Regularization

As seen before, model's generalization capability was a key factor to improve during the competition. Considering that only a 13% of the test data was available, the score obtained from it was just a small guidance and the private test set could contain unseen patterns that could lead to low scores.

As a way to prepare better a model to cover new observations, reducing any specificity learned from the training data and creating more generalized knowledge is achieved through the process of regularization.

A technique that perfectly fits the Multi Head BERT model's architecture was adapted from a proposal found in [51]. Multi-sample dropout suggests to apply multiple parallel dropout layers to replicas of a network layer (duplicating its structure), to then average the losses of them and optimize the shared weights. In this case, each of BERT's output heads will be considered those replicated layers and will be applied a dropout step before mixing them together.

5.1.3.1 Test

The designed ANN architecture for this test can be described as follow:

1. Input layers with token embeddings (WordPiece tokenized text), segment embeddings and position embeddings for the two text pairs
2. Two BERT layers using the last four heads as output for each text pairs
3. Multi-sample dropout layer for each of the eight layers
4. Concatenation of outputs from 3
5. Averaged pooled layer from 4

6. Linear layer of size 6144, compressing into size 1536 with ReLU activation
7. Concatenation of categorical features with layer from 5
8. Dropout layer from 7
9. Linear layer compressing into size 384 with ReLU activation
10. Linear combination and Sigmoid activation

The used loss function was *BCE* and the optimizer selected was *AdamW*. The configuration used to train the model had the following hyper-parameters:

- *Learning rate*: 3e-5
- *Batch size*: 5
- *Number of epochs*: 5
- *Dropout*: 0.2
- *Multi-sample Dropout*: 0.5
- *Computation method*: GPU (total runtime: 210 minutes)

5.1.3.2 Results

The metrics obtained from this model can be seen in Figure 5.4, and as expected the performance is similar to the one obtained in the previous model. Although peak scoring levels are similar, this model has the particularity of performing slightly faster and achieving the best score in less epochs across all folds.

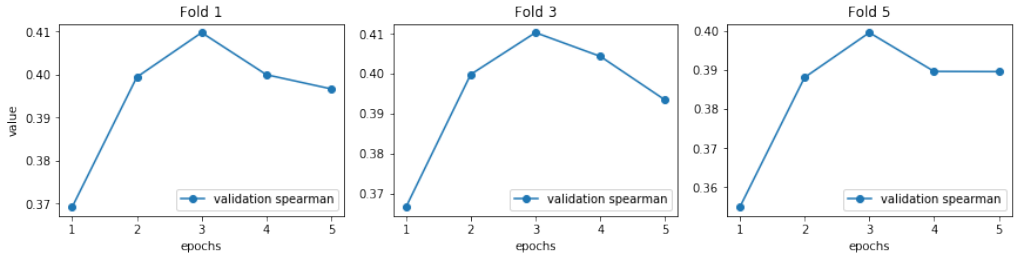


Figure 5.4: BERT Multisample Dropout model metrics

In order to evaluate properly if the regularization had a positive effect on the final submission, public and private scores are calculated for the comparison as seen in Table 5.4. While there are not big changes, the average 1% difference, lower for validation but higher for the rest, plus the faster convergence is a welcome improvement to the model.

Model	Validation score	Public test score	Private test score
Multi Head BERT	0.40959	0.39460	0.37243
Multi-drop BERT	0.40652	0.39767	0.37472

Table 5.4: BERT Multisample Dropout model final scores

Other regularization techniques were tried without success: variation on the L2 regularization lambda gave similar results, while adding batch normalization in between

BERT output layers and the MLP layer decreased performance *-31.19%*, which can be expected due to the fact that BERT already counts with batch normalization on its inner architecture and their outputs are susceptible to any type of whitening process.

5.1.4 Miscellaneous

Other minor tests and adjustments were performed and decided through performance comparison on the Spearman score on the first epoch for each fold or on longer runs in case there were doubts if the tweak could improve on the long run (which is limited to three epochs).

On the side of the successful tests, it is possible to mention:

- Weighted loss for unbalanced labels: *+0.4%*
- Using SELU activation instead of ReLU: *+1.76%*
- Using AdamW as optimizer with linear decay helped the model converge faster
- A learning rate of 3e-5 gave the most stable results in comparison with others in the range [1e-4, 5e-5]
- Further lowering the batch size only slowed model training, without giving improvements. The maximum batch size allowed by memory limitations was 5

Things that were tried and did not offer any improvements were:

- Text preprocessing techniques did not offer improvements. Performing text clean up, normalization, lemmatization or replacing contractions always decreased performance. Considering the WordPiece vocabulary and that is actually thought to work with raw text, this can be expected. The best scenario was achieved with mild cleanup and expanding contractions with -2.72%
- Novel optimizer Stochastic Weight Averaging (SWA [52]) was tested, and due to only training for a small number of epochs it did not manage to offer any improvements as it relays on longer iterations
- Learning rate manipulation with the CyclicLR [53] scheduler also gave worse results, due to training with a small dataset and having a small quantity of steps
- Trying to redesign the problem into a full classification one with a Softmax activation and predict the actual value of the label (and not the probability of being 1 like BCE) caused a decrease of: -3.64%
- Using a Sigmoid activation but optimizing through Huber loss gave a decrease of: -4.64%
- A novel proposal suggests a loss function capable of creating surrogates to optimize non-differentiable problems such as ordering. Unfortunately, SoDeep [54] is not well documented, and only the default model was tested, worsening performance by -35.45%

5.1.5 Postprocess

The resulting model still suffers from the impact of unbalanced labels as its detailed score seen in Table 5.5 displays the same trend previous models had.

Best performing fold	
question_asker_intent_understanding	0.4242
question_body_critical	0.6725
question_conversational	0.3687
question_expect_short_answer	0.3036
question_fact_seeking	0.3511
question_has_commonly_accepted_answer	0.4535
question_interestingness_others	0.3553
question_interestingness_self	0.4877
question_multi_intent	0.5919
question_not_really_a_question	0.1232
question_opinion_seeking	0.4476
question_type_choice	0.7567
question_type_compare	0.3773
question_type_consequence	0.1398
question_type_definition	0.3860
question_type_entity	0.5315
question_type_instructions	0.8007
question_type_procedure	0.3656
question_type_reason_explanation	0.6646
question_type_spelling	0.0399
question_well_written	0.4905
answer_helpful	0.2316
answer_level_of_information	0.4380
answer_plausible	0.1264
answer_relevance	0.1332
answer_satisfaction	0.2900
answer_type_instructions	0.7888
answer_type_procedure	0.3028
answer_type_reason_explanation	0.6636
answer_well_written	0.2020

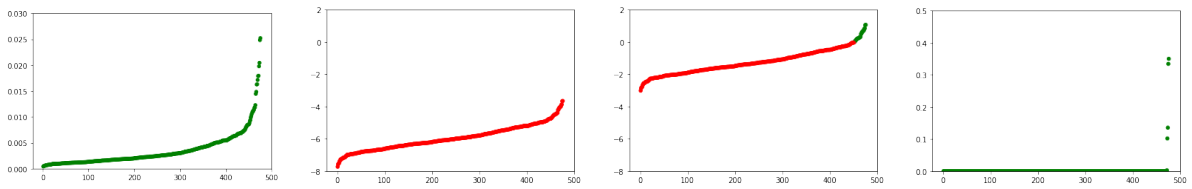
Table 5.5: Labels' Spearman's ρ performance

In order to partially overcome this negative behavior and help the model score better on labels where it could not learn properly how to predict, a post-processing technique was applied only for competition purposes.

Grabbing as an example the label *question_not_really_a_question* in training set it only has 66 samples with a value above 0 (1% of the total), meaning that 99% of the Q&A have the same ranking (tied at 0). This type of distribution has clearly proven difficult to predict for the model.

If considering the predicted values from a Sigmoid activation for this label, the nature of the function will produce a distribution where all the observation will have a different ranking (as seen in Figure 5.5a).

If instead the linear values are kept without Sigmoid activation (Figure 5.5b this will show a distribution of scores (in the case of this label with all of them below 0); that can be shifted (only if no positive values are present) to make at least a top percentage of them positive, as seen in Figure 5.5c; which allows for a ReLU activation that will



(a) Sigmoid (b) Linear (c) Shifted (d) Shifted+ReLU

Figure 5.5: Post-process for predicted scores of *question_not_really_a_question*

make everything below 0 to have the same ranking. The values from Figure 5.5 represent the correction to only the top 1% (creating only 5 out of 476 test Q&A with predicted positive values), and can be achieved with the following function:

```
from torch.functional.F import relu
from torch import Tensor
import numpy as np

percentile = 99
predicted_values = sample_prediction[:, column]

center_k = abs(np.percentile(predicted_values, percentile))
postprocess_values = np.array(relu(Tensor(np.array(predicted_values + center_k))))
```

To further improve final scoring, the following unbalanced labels where adjusted to their distributions: *question_not_really_a_question, question_type_consequence, question_type_procedure, question_type_spelling, question_conversational, question_type_compare, question_type_definition, question_type_entity, question_fact_seeking, question_has_commonly_accepted_answer, answer_helpful, answer_relevance, answer_type_procedure, answer_plausible*

Once this was done, values for all 30 labels were transformed into rankings and normalized into a scale from 0 to 1. Lastly this post-processed predictions (one for each folds of the model) are averaged. This process can be performed by the following piece of code:

```
def ranking_blend(predictions, target_columns):
    ranked_val = np.array(predictions)
    for col in range(len(target_columns)):
        ranked_val[:,col] = rankdata(np.array(predictions[:,col]))
        ranked_val[:,col] = ranked_val[:,col] / (len(ranked_val[:,col]) + 1)
    return ranked_val

final_predictions = np.average([ranking_blend(fold_prediction, target_columns)
                                    for fold_prediction in model_predictions
                                ], axis=0)
```

5.2 Final model

The final model created [55] was the combination of the previous successful tests with the verification that there was no conflicting behaviour between them.

In order to obtain the best model possible, training is performed over a 10 fold split of the dataset and, based on previous findings, only three epochs will be needed as the model starts overfitting afterwards.

5.2.1 Model

The final ANN architecture can be described as follow:

1. Input layers with token embeddings (WordPiece tokenized text), segment embeddings and position embeddings for the two text pairs
2. Two BERT layers using the last four heads as output for each text pairs
3. Multi-sample dropout layer for each of the eight layers
4. Concatenation of outputs from 3
5. Averaged pooled layer from 4
6. Linear layer of size 6144, compressing into size 1536 with SELU activation
7. Concatenation of categorical features with layer from 5
8. Dropout layer from 7
9. Linear layer compressing into size 384 with SELU activation
10. Linear combination for output

The used loss function was a *weighted BCE* and the optimizer selected was *AdamW* with customized decay. The configuration used to train the model had the following hyper-parameters:

- *Learning rate*: 3e-5
- *Folds*: 10
- *Batch size*: 5
- *Number of epochs*: 3
- *Dropout*: 0.2
- *Multi-sample Dropout*: 0.5
- *Computation method*: GPU (total runtime 450)

5.2.2 Results

Since the goal is to achieve maximum performance with an inference time below 2 hours, the best combination of epochs from each fold were used for inference. The best performing model used for inference only the second epoch of each fold. The submission results for the model with and without post-processing can be seen in Table 5.6:

Model	Public test score	Private test score
Final BERT	0.40861	0.38543
Final BERT+Post Process	0.41800	0.39303

Table 5.6: Final model scores

Both models where selected for final submission, with the post-processed version resulting 92nd of 1571 competitors (top scoring 6%) and gaining a Bronze medal,

while the the non post processed version would have also obtained a Bronze medal at the 114th position. Note in Figure 5.6 how the model increased 34 places in comparison with the scoreboard from the public test set, this boost was obtained thanks to the model being better at generalization (due to the regularization technique and the 10 fold training) than the others on this range.

Figure 5.6: Kaggle competition scoreboard [1]

Chapter 6

Conclusion

This final chapter offers some closing thoughts about the work done, the results obtained and the status of the NLP field.

6.1 Conclusions

In spite of the fact that the model did not achieved top performance, it is important to mention that the goal was always to push BERT to its limit and obtain the best performing single model while also following a research standard capable of give insights on BERT's optimization. If this is taken into account, plus the fact that the model at least managed to rank into medal range of the competition, we can consider this work as successful.

All top scoring competitors heavily relied on the use of BERT and a combination of several related models like XLNET, RoBERTa, BART, ULMFiT or GPT2 [1]. This multi-model ensemble technique is a traditional method used in Kaggle competitions to try to get the best score possible, but its usefulness is often far in terms of scientific validity or general industrial application. The 1st place of the private test set achieved a score of 0.43100 with a multi model ensemble and heavy post-processing, while the first place of the public set (which had an impressive score of 0.48967) fell into the sixth place of the private set with a score of 0.42705, showing how the model relied on its success through overfitting and post-processing based on the training set.

A curiosity from this challenge is the fact of how even top models did not manage to score high or at least above the 0.5 mark in neither of the test sets. An easy guess can be that the problem itself is too complex, but with some consideration of the general results and some of the findings from this research, it can be said that this low performance could happen due to a mix of two reasons: dataset quality and algorithm limitations.

As Google mentions on the challenge proposal, this is a new problem and for it they manually created a new dataset. This data was not created organically, nor obtained automatically, it was labeled manually by a group of less than 100 people. As part of being such a small number, the resulting labels can contain biases or at least, lack of a real "common-sense" pattern. Even further, manually studying several of the unbalanced Q&A raises some questions about the real purpose of some labels

(e.g. *question_type_spelling*), and the fact that Kaggle has not released the full test set, makes it even harder to analyze properly the real value of this dataset.

On the other hand, there is no real algorithm developed for the problem of giving common-sense scores to text. This cognitive process is still one of the challenges of complete AI and perfectly mapping our decision making process through subjectivity and common-sense into a mathematical formula seems to still be far from our time. Nonetheless, current state of the art models prepared for different tasks still managed to perform decently with some tweaking. If the top (i.e. easy) labels are the only ones considered (e.g. *question_body_critical*) it is interesting to see how a model designed to do Q&A adjusts itself into finding patterns in the text that explain a certain label. Unfortunately, the labels used on this dataset were too broad and there were not enough observations to work with, thus creating a model capable of understanding all these "common-sense" aspects at the same time will be hard to find.

While some of the labels were not designed in a way that fits into the logic and capacity of Transformers or Attention (e.g. *question_not_really_a_question, answer_well_written*), since they rely in text components that fall into more abstract concepts, a fine-tuned BERT is still capable of improving the performance of hand crafted models aiming for those specific concepts. This achievement certainly shows the real power of transfer learning models and how finding ways to reuse general knowledge embedded in ANN can lead to more future improvements.

With all this being said, it is possible to be certain that Google used this competition to test the real value of their new dataset and to encourage people to push BERT to its limits (several statements on the competition's forum assert that from all state of the art models at that time, BERT was the best performer). A blind tendency of these times is to build bigger pre-trained models with much more data and more time for training, leading to better state of the art benchmarks at the cost of more computational cost, without caring about its resources cost or the efficiency they could reach through proper optimization.

The evolution of the field of NLP, as covered by Chapters 2 and 4, shows how complex and interesting this research area has gotten over the years. As a first approach into NLP, the learning curve can start slow as old linguistic based approaches are covered, but once numerical representation of text and Deep Learning gets added into the field, the learning curve becomes very steep as multiple complex concepts bring a series of previous knowledge as requirement. Is because of this complexity, the level of innovation needed and the challenging problems that make NLP one of the most exciting areas of research on AI nowadays.

6.2 Future work

Same as most of Deep Learning models, BERT can be considered as a black box algorithm. Even at the moment of publication, their creators were not able to fully justify some of its behaviours and subsequent papers focused on explaining different parts and inner functioning. A way to improve this research could come from performing ablation studies to inspect and understand better how each component of BERT and the parts of the proposed final ANN model work. Also, probing has proven to be useful [56] to understand the inner processes of BERT.

In terms of future improvements, exploring the performance obtained by tweaking some of the models that came after BERT, such as: RoBERTa, XLNET or DeepSpeed could be fruitful or, at least, useful for a model ensemble scenario (a RoBERTa baseline model was tested during experimentation but its performance did not encourage to keep exploring it).

Also, using distilled versions like AlBERT or DistillBERT, could allow the creation of bigger models with the same computation resources and eventually lead to better performance (DistillBERT was tested during the research but did not seem to give better results for this dataset on an initial run).

Other possible improvements could come from data augmentation (creating enriched text features through FastText or ontologies), or redesigning the dataset in a way that can express the ordering problem to be solved (i.e. TripletLoss).

References

[1] Kaggle. (2019). Google QUEST Q&A Labeling, [Online]. Available: https://www.kaggle.com/c/google-quest-challenge/overview (visited on 03/2020).

[2] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding", *arXiv preprint arXiv:1804.07461*, 2018.

[3] V. Lopez, M. Pasin, and E. Motta, "Aqualog: An ontology-portable question answering system for the semantic web", pp. 546–562, 2005.

[4] W.-t. Yih, M.-W. Chang, X. He, and J. Gao, "Semantic parsing via staged query graph generation: Question answering with knowledge base", pp. 1321–1331, Jul. 2015. DOI: 10.3115/v1/P15-1128. [Online]. Available: https://www.aclweb.org/anthology/P15-1128.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", 2018. arXiv: 1810.04805 [cs.CL].

[6] P. Nayak. (2019). Understanding searches better than ever before, [Online]. Available: https://www.blog.google/products/search/search-language-understanding-bert/ (visited on 03/2020).

[7] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?", pp. 194–206, 2019.

[8] M. Ostendorff, P. Bourgonje, M. Berger, J. Moreno-Schneider, G. Rehm, and B. Gipp, "Enriching bert with knowledge graph embeddings for document classification", *arXiv preprint arXiv:1909.08402*, 2019.

[9] Kaggle. (2019). Google QUEST Q&A Labeling, [Online]. Available: https://www.kaggle.com/c/google-quest-challenge/data (visited on 03/2020).

[10] P. University. (2010). WordNet, [Online]. Available: https://wordnet.princeton.edu/ (visited on 03/2020).

[11] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[12] D. Rosenberg, "Stop, words", *Representations*, vol. 127, no. 1, pp. 83–92, 2014.

[13] B. Santorini, "Part-of-speech tagging guidelines for the penn treebank project (3rd revision, 2nd printing)", *Ms., Department of Linguistics, UPenn. Philadelphia, PA*, 1990.

[14] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press, 2008.

[15] A. I. Kadhim, Y.-N. Cheah, I. A. Hieder, and R. A. Ali, "Improving tf-idf with singular value decomposition (svd) for feature extraction on twitter", pp. 144–152, 2017.

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[17] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks", pp. 971–980, 2017.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[19] M. D. Zeiler, "Adadelta: An adaptive learning rate method", *arXiv preprint arXiv:1212.5701*, 2012.

[20] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond", *arXiv preprint arXiv:1904.09237*, 2019.

[21] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space", *arXiv preprint arXiv:1301.3781*, 2013.

[22] D. Sarkar. (2018). A hands-on intuitive approach to Deep Learning Methods for Text Data — Word2Vec, GloVe and FastText, [Online]. Available: https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c44370bbfa (visited on 03/2020).

[23] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification", 2016. arXiv: 1607.01759 [cs.CL].

[24] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation", pp. 1532–1543, 2014. [Online]. Available: http://www.aclweb.org/anthology/D14-1162.

[25] Y. Goldberg, "Neural network methods for natural language processing", *Synthesis Lectures on Human Language Technologies*, vol. 10, no. 1, pp. 1–309, 2017.

[26] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations", *arXiv preprint arXiv:1802.05365*, 2018.

[27] C. Olah and S. Carter, "Attention and augmented recurrent neural networks", *Distill*, 2016. DOI: 10.23915/distill.00001. [Online]. Available: http://distill.pub/2016/augmented-rnns.

[28] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation", 2015. arXiv: 1508.04025 [cs.CL].

[29] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization", 2017. arXiv: 1705.04304 [cs.CL].

[30] J. Cheng, L. Dong, and M. Lapata, *Long short-term memory-networks for machine reading*, 2016. arXiv: 1601.06733 [cs.CL].

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need", pp. 5998–6008, 2017.

[32] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, *et al.*, "Universal sentence encoder", *arXiv preprint arXiv:1803.11175*, 2018.

[33] H. Face. (2020). BERT Documentation, [Online]. Available: https://huggingface.co/transformers/model_doc/bert.html (visited on 03/2020).

[34] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing", *ArXiv*, vol. abs/1910.03771, 2019.

[35] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations", 2019. arXiv: 1909.11942 [cs.CL].

[36] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter", 2019. arXiv: 1910.01108 [cs.CL].

[37] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding", 2019. arXiv: 1906.08237 [cs.CL].

[38] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context", 2019. arXiv: 1901.02860 [cs.LG].

[39] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach", 2019. arXiv: 1907.11692 [cs.CL].

[40] Kaggle. (2020). Kaggle Notebooks, [Online]. Available: https://www.kaggle.com/notebooks (visited on 03/2020).

[41] Google. (2020). Google Colaboratory, [Online]. Available: https://colab.research.google.com/ (visited on 03/2020).

[42] TensorFlow. (2020). TensorFlow Keras BERT Base Uncased, [Online]. Available: https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1 (visited on 03/2020).

[43] HuggingFace. (2020). PyTorch BERT Base Uncased, [Online]. Available: https://github.com/huggingface/transformers/blob/master/src/transformers/modeling_bert.py#L36 (visited on 03/2020).

[44] Google. (2020). BERT vocabulary, [Online]. Available: https://github.com/huggingface/transformers/blob/master/src/transformers/tokenization_bert.py#L35 (visited on 03/2020).

[45] Wikipedia. (Mar. 2020). Spearman's rank correlation coefficient, [Online]. Available: https://en.wikipedia.org/wiki/Spearman's_rank_correlation_coefficient.

[46] A. Fabiano. (2020). BERT Classifier Model Notebook, [Online]. Available: https://github.com/arielfabiano/googlequest-challenge/blob/master/google-quest-bert-classifier.ipynb (visited on 03/2020).

[47] ——, (2020). LSTM Model Notebook, [Online]. Available: https://github.com/arielfabiano/googlequest-challenge/blob/master/google-quest-lstm-model.ipynb (visited on 03/2020).

[48] ——, (2020). USE Model Notebook, [Online]. Available: https://github.com/arielfabiano/googlequest-challenge/blob/master/google-quest-use-model.ipynb (visited on 03/2020).

[49] ——, (2020). BERT Classifier Model Notebook, [Online]. Available: https://github.com/arielfabiano/googlequest-challenge/blob/master/google-quest-bert-classifier.ipynb (visited on 03/2020).

[50] ——, (2020). BERT Sequence Model Notebook, [Online]. Available: https://github.com/arielfabiano/googlequest-challenge/blob/master/google-quest-bert-sequence.ipynb (visited on 03/2020).

[51] H. Inoue, "Multi-sample dropout for accelerated training and better generalization", 2019. arXiv: 1905.09788 [cs.NE].

[52] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, "Averaging weights leads to wider optima and better generalization", 2018. arXiv: 1803.05407 [cs.LG].

[53] L. N. Smith, "Cyclical learning rates for training neural networks", 2015. arXiv: 1506.01186 [cs.CV].

[54] M. Engilberge, L. Chevallier, P. Pérez, and M. Cord, "Sodeep: A sorting deep net to learn ranking loss surrogates", 2019. arXiv: 1904.04272 [cs.LG].

[55] A. Fabiano. (2020). BERT Final Model Notebook, [Online]. Available: https://github.com/arielfabiano/googlequest-challenge/blob/master/Google_Quest_PyTorchBERT_finetune_final.ipynb (visited on 03/2020).

[56] T. Niven and H.-Y. Kao, "Probing neural network comprehension of natural language arguments", 2019. arXiv: 1907.07355 [cs.CL].