Siamese Transformer Networks for Key-Point Analysis

Mariani Valerio (560014) - v.mariani5@studenti.unipi.it
Marinelli Alberto Roberto (638283) - a.marinelli9@studenti.unipi.it
Master Degree Computer Science, AI curriculum
HLT course (654AA), Academic Year: 2022-2023

Abstract

Pretrained language models are nowadays becoming standard approaches to tackle various Natural Language Processing tasks. This is the reason why we decided to experiment with using Transformers in a **Siamese network** to solve these problems and understand how they work. Specifically, due to the extensive pre-training of the available language models and the small dataset, a pre-trained model of **BERT** and its variant **RoBERTa** inside a Siamese network were chosen. In particular, this architecture was used to tackle task 1 in the **IBM's shared task 2021**. In this task, we are given a set of debatable topics, a set of key points, and a set of arguments, supporting or contesting the topic. We are then asked to match each argument to the topic it is supporting or contesting.

Contents

1 Introduction

The aim of this project is to implement and explore architectures based on **Siamese Transformer networks** to solve Natural Language Processing tasks, in particular the one proposed by **IBM's shared task 2021** analysed in chapter 2, where the specifications of the problem addressed and the analysis of the dataset are described.

Subsequently, Chapter 3 provides a description of the theoretical concepts required to understand the functioning of the models used, in particular a focus was made on transformers such as BERT and its variant RoBERTa, Siamese networks and how these two concepts were combined to create Sentence BERT.

In Chapter 4, details of the implementation are given and then in Chapter 5, the results obtained are reported.

Finally, the last Chapter (6) is dedicated to conclusions and considerations on the results obtained and possible future developments.

2 Problem description (competition)

Before looking at the network architecture in detail, it is necessary to better understand the context in which they were used. This section provides the specifications of IBM's shared task 2021 and then an analysis of the dataset.

2.1 Task

The task of key-point analysis requires an understanding of natural language that is powerful enough to extract the semantic meaning from a sentence and condense it into a different sentence: one that has the same meaning but is more concise than the first one. We call these two sentences an argument and a key point. As described in [1], the task here is to score the ability of each key point to represent a summary of an argument.As stated in [2] indeed, it is often desirable not only to extract the most salient points, but also to quantify their prevalence.

In our case, we are tackling track 1 of the the ArgKP-2021 shared task, in which we are given a debatable topic, a set of key points per stance, and a set of crowd arguments supporting or contesting the topic, and we need to report, for each argument, its match score for each of the key points under the same stance towards the topic.

2.2 Dataset analysis

The ArgKP dataset [1] contains about ∼24,000 topic/keypoint pairs for 28 controversial topics and is divided into training and development sets. Each of the pairs is labeled as matching / not matching and assigned a position towards the topic. Given a set of key points for a topic, a topic can be matched to one or more key points or to none of them.

In order to get a more detailed view of the data properties, operations were performed on the dataset. Initially, the distribution of positive and negative examples in the training set and dev set was analyzed and shown on a barplot.

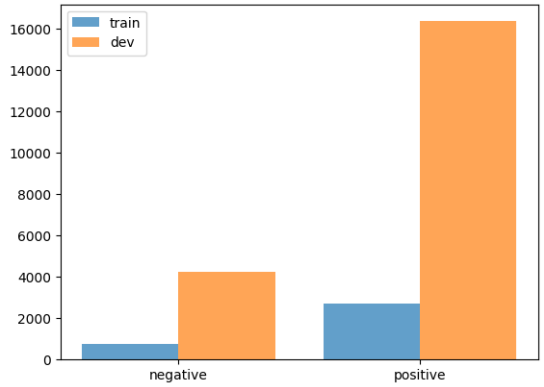

Figure (1) Barplot positive and negative distribution.

As we can see from Figure 1 the negative examples, in both datasets, are almost three times as many as the positive examples.

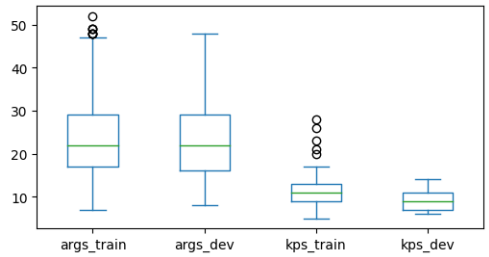

Figure (2) Boxplot lengths of tokens.

Then, the maximum and average tokenized input lengths of arguments and keypoints (details in Table 1) were analyzed in order to estimate the input length to be supported by the models chosen to solve the task.

	count	mean lenght	standard dev	min	25%	50%	75%	max
arg (train)	5583.0	23.322228	8.598840	7.0	17.0	22.0	29.0	52.0
arg (dev)	932.0	23.185622	8.603255	8.0	16.0	22.0	29.0	48.0
kps (train)	207.0	11.202899	3.580881	5.0	9.0	11.0	13.0	28.0
kps (dev)	36.0	9.111111	2.411612	6.0	7.0	9.0	11.0	14.0

Table (1) Statistics on lenght of sentences.

Finally, we analyzed the number of keypoints for each topic to asses the probability of correctly classifying each sample with a random assignment. The result is about 25%.

3 Models

In this section, we provide an overview of the main theoretical concepts of the models we used to conduct our experiments. In particular, we start by introducing the BERT and RoBERTa architectures [5, 6], and the concept of Siamese Networks [3]. Then, we mix up these topics to introduce a neural model known as Sentence BERT (SBERT) that was introduced in [7].

3.1 BERT

BERT (*Bidirectional Encoder Representations from Transformers*) is an open source machine learning framework for natural language processing (NLP). It is based on the transformers models, in which each output element is connected to each input element and the weights between them, using the attention mechanisms, are dynamically determined based on their connection. BERT has been pre-trained on the BooksCorpus and Wikipedia Corpus, as well as two separate NLP tasks: Masked Language Modeling and Next Sentence Prediction.

3.1.1 BERT Architecture

The **architecture of BERT** is based on multi-layer bidirectional Transformer [8] that exploit attention mechanisms to extract relationships between words in a sentence. In detail, BERT consists of an encoder with the aim of generating a language representation model, so it does not use the decoder part of classical Transformers.

There are several pre-trained models, in our specific case we used the **BERT BASE** model, consisting of 12 Transformer blocks, 12 Attention heads and 768 hidden layer size.

3.1.2 Input Representation

In order for texts to be processed, they must be converted into a sequence of tokens (one for each word in the input text) to which the following extra information are added:

1. **Special tokens**: *token [CLS]* is added as the first token in the input, *token [SEP]* is used to separate each sentence within the same input.

2. **Segment embedding**: marker added for each token in order to distinguish them for each sentence.

3. **Position embeddings**: a positional embedding is done to each token in order to identify its position within the sentence.

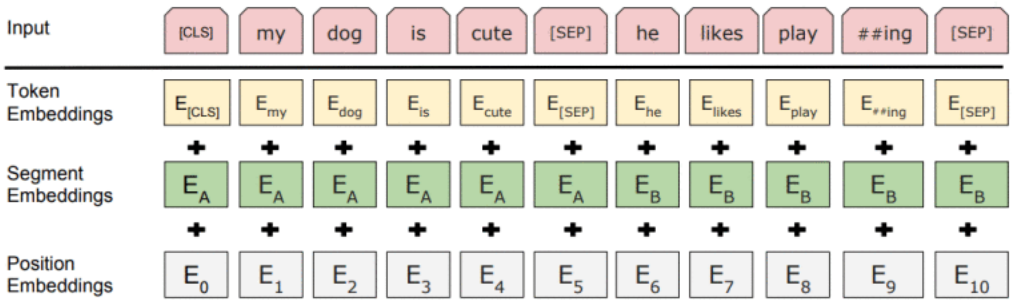

Figure (3) BERT input representation.

3.1.3 Pre-training BERT

BERT is pre-trained on two unsupervised tasks: **Masked LM** and **Next Sentence Prediction (NSP)**.

In the first task, 15% of the input words are masked through a special token ([MASK]); the final goal is to predict these words using only the context in which they occur, and so the unmasked words.

In the second task, on the other hand, the goal is to understand the relationship between two sentences. During training, the model receives pairs of sentences as input and learns to predict whether the second sentence is also the next sentence in the original text. The input dataset consists of 50% related sentences and the remaining 50% random sentence pairs.

3.2 RoBERTa

Numerous variants have been developed from the BERT model, in this project we focused on **RoBERTa** [6].
RoBERTa, which stands for *'A Robustly Optimised BERT Pretraining Approach'*, has only been pre-trained with the MLM (Masked Language Modeling) objective on the combination of five datasets (BookCorpus, English Wikipedia, CC-News, OpenWebText and a subset of CommonCrawl data) with a total weight of 160GB.
This model shares most of the properties of the BERT architecture, the only differences being:

1. It uses a byte-level BPE tokenizer with a larger subword vocabulary (50k compared to BERT's 32k)

2. Implements dynamic word masking and abandons the next sentence prediction task.

3. RoBERTa's training hyperparameters.

3.3 Siamese Networks

The concept of Siamese Network was first introduced by Yan Lecunn in [3], and was described there as a neural architecture with two sub-networks, joined at the output with the aim of making the two sub-networks produce similar representations for semantically similar samples. In that case, the authors wanted to implement signature verification, by **minimizing the (cosine) distance between representations of signatures from the same author**, while at the same time maximizing the distance between representations of signature made by two different authors. All of this was done with the constraint that **the two sub-networks must have identical weights** on a training/validation dataset in which each is in the form: $\{(s1, s2) \rightarrow |\cdot|_{cos} \in [-1, 1]\}$, where $|\cdot|_{cos}$ is cosine similarity, meaning that they had **set of couples of samples, annotated with 1 if the two samples were semantically equivalent, -1 otherwise**. A graphical representation of the working principle of the Siamese Network in [3] is shown in Figure 4.

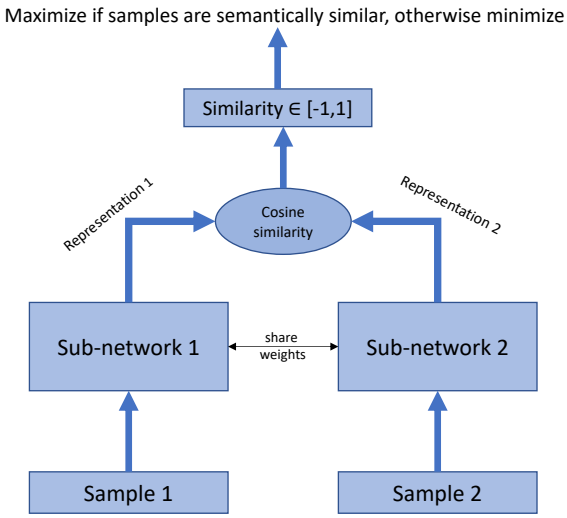

Figure (4) Schema of high level architecture of a siamese network

In recent times, Siamese Networks have been gaining in popularity and have been deeply studied in the context of Self-Supervised Learning by means of their relationships with Energy-Based Models (EBM) [4]. An EBM is a model that can be trained to evaluate the compatibility of two samples with each other, stated as value of a single output value known as energy. The highest the energy produced by the EBM, the

stronger the hypostesis that the two samples incompatible. In this context, the Siamese model was extended by not constraining the two representations by means of cosine similarity, and allowing instead the use of any module that could give a measure of energy as described above, i.e. a Softmax layer that outputs the probability of the two samples being semantically similar.

3.4 Sentence BERT

Sentence BERT (SBERT) is an architecture proposed in [7] that combines the BERT model and the concept of Siamese Networks. The main idea is to **reduce computational cost of pairing sentences** by training a network to produce **sentence embeddings** that are (numerically) **similar for sets of sentences with semantically similar sentences**. This way, **instead of computing a label for every pair of sentences**, one produces **an embedding for each sentence, then labels as similar those sentences that have similar embeddings**. This also introduces the possibility for efficient, sentence-level, clustering and information retrieval.

3.4.1 Sentence Embeddings

The most common ways to produce sentence-level embeddings with BERT are:

1. using the CLS token's embedding. This is a special token that the BERT Tokenizer puts at the beginnig of every sentence and it's used for classification;

2. averaging on BERT's output layer's activations.

In the following sections, we refer to the first method as **CLS Strategy**, and the second as **AVG Strategy**. In practice, those embeddings are vectors of numbers, and the aim of the training procedure is to learn those sentence embeddings in such a way that they better capture the semantical meaning of the sentence.

3.4.2 Training Procedures

On top of the two sentence embeddings strategies, the authors in [7] propose two training procedures, based on two Siamese architectures. The way the architecure is in the "joining" module, that is connected to the two BERT networks: they 1) join the sentence embeddings by means of a **(softmax) classification layer** connected to the concatenation of the two embeddings than train with classification objective function; 2) join the embeddings by means of a **custom layer that computes cosine similarity** between the two embeddings and then train with a regression objective function.

4 Implementation

This section explains the implementation phase and the organisation of the project files containing all the necessary functions.

Within the file `SBERT_models.py` are all the functions required to build Siamese network models. In particular, with the '*build_model*' function it is possible to assemble different types of models by passing as input a '*config*' dictionary (for selecting the main training parameters such as learning rate or number of epochs and choosing the strategy to be used to process the output of the transformer such as CLS or AVG), with the '*embeddings*' parameter you can select the transformer within the network (BERT or RoBERTa), the '*join*' parameter allows you to define whether to use the model with softmax or cosine similarity, finally, you can also define an optional '*schedule*' for training (such as linear decay or warmup cosine decay, whose functions are defined within the same file).

More specifically, we took pre-trained **BERT** and **RoBERTa** models from huggingface (aggiungere link qui) and then composed our Siamese models with python's framework **keras-tensorflow**. The choice was guided by the great readability and clarity that these frameworks offer when it comes to writing Deep Learning models.

Then we needed to get, from the pre-trained models, embdeddings of both the arguments and the key-points, so we passed them the tokenized input:

```
encode_input = TFBertModel.from_pretrained('bert-base')
out1 = encode_input(tokenized_args_train, ...)
out2 = encode_input(tokenized_kps_train, ...)
```

At this point, we had the total output of the BERT/ RoBERTa models and we needed to extract sentence-level embeddings. As stated in section 3.4.1, we had two strategies to compose the output into a single vector: the **CLS Strategy** and the **AVG Strategy**. The CLS was performed by taking the first vector of the transformer's output (derived from the CLS token which is always at the beginning of the sentence), instead, the AVG strategy is calculated by averaging the entire output of the transformer.

Finally, we joined the two sentece embeddings. For the **cosine similarity joint layer**, we implemented a custom layer that computes cosine similarity, like so:

```
cosine_similarity = tf.keras.layers.Dot(axes=1,normalize=True)
preds = cosine_similarity([emb1,emb2])
```

And got a single number $\in [-1, 1]$ that we could we use with a regession objective function.

For the **softmax joint layer** instead, we wrote the following code:

```
sub_emb = tf.keras.layers.subtract([emb1, emb2])
concat = tf.keras.layers.Concatenate(axis=-1)([emb1, emb2, sub_emb])
preds = tf.keras.layers.Dense(2, activation="softmax")(concat)
```

And got a two-way softmax output, to be used with binary cross-entropy classification objective function.

5 Experiments

During the testing phase, we focused on architectures based on BERT (base) and RoBERTa as transformers and with cosine similarity to process their output.

As mentioned in section 3.1.2, it is necessary to process the datasets (training and validation) to make them usable with the above-mentioned transformers. To do this, we used the tokenizers provided directly by the library, which automatically converts the input into tokens, adds the special tokens and padding, and generates the attention masks.
In example, for the RoBERTa-based model, we started by taking **RoBERTa tokenizers** from huggingface and passed them the row text, like so:

```
0    from transformers import RobertaTokenizer
1    tokenizer =
2    RobertaTokenizer.from_pretrained('roberta-base')
3    tokenized_args_train = tokenizer(args_train,...)
```

The same was done for the BERT-based Network.

Afterwards, we built Dataset type objects (provided by the Tensorflow library) to perform readable and efficient shuffling and batching of samples. Another benefit of tensorflow Dataset class is the zip method to provide training samples in the form described in section 3.3 :$\{(s1, s2) \rightarrow |\cdot|_{cos} \in [-1, 1]\}$, where $|\cdot|_{cos}$ is desired cosine similarity, while in this case $s1$ is an argument, and $s2$ a possible key-point.

For each of the two architectures (both BERT-based and RoBERTa-based), the search phase is divided into two. Initially, we ran an initial grid search on a total of $\sim$80 configurations to find interesting intervals in hyperparaters, then we tested the best intervals on 5 dataset initialisations and averaged the results. Each training was carried out with a batch size of 16.

5.1 Experimental Setting

As described in section 2.2, our dataset is very unbalanced (positive samples are $\sim 1/3$ of the negatives ones). Instead of downsampling the dataset (that is already small) or usampling by repetition, we decided to tune the cosine similarity as hyperparameter. In more, it didn't seem right to say that if two sentences don't match, correct cosine similarity is -1 (that means opposite semantical meaning).

After some preliminary trials, we obtained the best results by labeling negative samples with 0.5 and positive samples with 1 (cosine similarity).

5.2 Evaluations

5.2.1 Evaluation Metric

The metrics for evaluation of the models on the development set were actually given in the task's repository. Each argument can be matched with a certain number of key points, so for each of them, a matching score must be given. Then, each argument is paired with the highest scoring key point assigned to it. After this first step, to make the evaluation more robust, only 50% highest-scoring couples is kept, while the others are discarded. Finally, the evaluation metrics on this subset of the results are Mean Average Precision (MAP) strict and relaxed. In our case, those values are always equal to each other because we assign a similarity measure to each couple instead of leaving some labels to be automatically filled by the evaluation metrics (as may be the case using other approaches).

As for average precision, the formula used is :

$$AP = \sum (R_n - R_{n-1}) P_n \tag{1}$$

where P_n and R_n are the precision and recall at the nth threshold. Average precision is computed for each topic, then AP values are averaged among the topics.

5.2.2 Evaluation Results

All results stated in the following are computed on the development set from the original task, with the above-mentioned metrics.

At the very beginning, we manually searched the space of hyper-parameters. In particular we tested different combinations of learning rates schedules: constant, linear decay and warm-up cosine decay. In the case of BERT, all of this was tested both on AVG and CLS Strategies. We decided to discard the CLS Strategy because it didn't produce better results than AVG (in accordance with the results in [7]). Some results were also tested on two/four epochs, almost always with more than one training epoch the model overfitted the data.

In addition, several configurations of architectures based on the use of Softmax were tested, but - due to the output format - these could not be subjected to evaluation using the metrics provided by the competition, as described in Chapter 5.2.2 and were therefore discarded. Thus, the models used are exclusively with cosine similarity.

Subsequently, the coarse-grained grid search was configured as in table 2. From there, we selected the best hyper-parameters for further experimentation.

Learning rate	MAP (SBERT)	MAP (SRoBERTa)
8e-06	0.733	0.805
(8e-06, 7e-06)	**0.760**	0.803
(8e-06, 6e-06)	0.742	0.775
(8e-06, 5e-06)	**0.767**	0.806
(8e-06, 1e-06)	0.749	**0.811**
7e-06	0.749	0.802
(7e-06, 6e-06)	0.746	0.801
(7e-06, 5e-06)	0.734	0.791
(7e-06, 1e-06)	0.728	0.807
6e-06	0.712	**0.821**
(6e-06, 5e-06)	0.759	0.790
(6e-06, 4e-06)	0.728	0.789
(6e-06, 1e-06)	0.749	**0.818**
5e-06	0.728	0.800
(5e-06, 4e-06)	0.757	0.794
(5e-06, 3e-06)	0.739	0.797
(5e-06, 1e-06)	0.753	0.795
4e-06	0.749	**0.814**
(4e-06, 3e-06)	0.747	0.792
(4e-06, 2e-06)	0.740	0.783
(4e-06, 1e-06)	0.742	0.779
3e-06	**0.770**	0.790
(3e-06, 2e-06)	0.759	0.773
(3e-06, 1e-06)	0.738	0.788
2e-06	0.749	0.754
(2e-06, 1e-06)	0.724	0.778
1e-06	0.706	0.770

Table (2) Coarse grained grid search. When two learning rates are reported, learning rate linearly decays from the first value to the second.

After the initial grid search, we focused on the best hyperparameter intervals, and we performed a finer-grained grid search to extract the best models. The results are listed in table 3 for SBERT and in table 4 for SRoBERTa.

Learning rate	MAP (SBERT)
(8e-6, 7e-6)	0.749 ± 0.009
(8e-06, 5e-06)	**0.753** ± 0.003
3e-6	0.747 ± 0.015

Table (3) SBERT - finer grained grid search. When two learning rates are reported, learning rate linearly decays from the first value to the second.

Learning rate	MAP (SRoBERTa)
(9e-6, 1e-6)	0.806 ± 0.004
8e-6	**0.817** ± 0.013
(8e-6, 1e-6)	0.806 ± 0.008
6e-6	0.802 ± 0.008
(6e-6, 1e-6)	0.793 ± 0.007

Table (4) SRoBERTa - finer grained grid search. When two learning rates are reported, learning rate linearly decays from the first value to the second.

6 Conclusion

Since the publication of the Tansformer Architecture, research in the field of Machine Learning is heavily concentrated on studying the capabilities of these models. For this project, we studied the ability of BERT and RoBERTA to be used in composition with a particular framework (Siamese Networks) that is very well suited to the task at hand (Key Point Analysis).

Despite the relatively small size of the Dataset we were given - expecially when compared to the corpus these models were trained on - both SBERT and Siamese-RoBERTa obtained some nice generalization of the ability to select the Key-Point of a sentence in a set of possible ones from the training set to the development (validation) set we were given. In fact, using the given metrics on the validation set, we obtained $\sim$75% Mean Average Precision with SBERT and $\sim$80% Mean Average Precision with Siamese-RoBERTA.

As for some possible future works, it could be noted for example that we focused more on models than on the improvement of the (relatively small and unbalanced) dataset on which they were trained, so it could be interesting to take the same models but try to augment the data - maybe also with other corpora - and see if they obtain better generalization results. In more, because of computational constraints, we were not able to try some more configuration, e.g. **bigger batch sizes** or larger models (BERT and RoBERTa large).

References

[1] Roy Bar-Haim et al. "From arguments to key points: Towards automatic argument summarization". In: *arXiv preprint arXiv:2005.01619* (2020).

[2] Roy Bar-Haim et al. "Quantitative argument summarization and beyond: Cross-domain key point analysis". In: *arXiv preprint arXiv:2010.05369* (2020).

[3] Jane Bromley et al. "Signature verification using a" siamese" time delay neural network". In: *Advances in neural information processing systems* 6 (1993).

[4] Sumit Chopra, Raia Hadsell, and Yann LeCun. "Learning a similarity metric discriminatively, with application to face verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 539–546.

[5] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[6] Yinhan Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[7] Nils Reimers and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks". In: *arXiv preprint arXiv:1908.10084* (2019).

[8] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).