Semantic sentence similarity

Daria Ozerova

***Abstract*—Semantic textual similarity is an important natural language processing task that measures the similarity of meaning between words, sentences, and documents. The state-of-the-art algorithms are based on the transformer architecture, typically represented by Bidirectional Encoder Representations from Transformers (BERT) and its derivatives. They achieve outstanding results, but classical BERT is extremely computationally intensive. So it could not be used in real applications. An architecture called bi-encoder solves this issue, making transformers usable for practical tasks. Both architectures are evaluated to provide fair results.**

The experiment part of this work evaluates a bi-encoder of a Robustly Optimized BERT Pre-training Approach (RoBERTa) on the STS Benchmark dataset. It is later shown how transformers behave when presented with challenging data, such as sentences with grammatical mistakes, typographical errors, and ambiguity. Finally, it is demonstrated how bi-encoders can be fine-tuned on a specially created domain-specific dataset. The final model is fine-tuned with just a few hundred examples. It achieves 20-25% better performance on a test set.

I. Introduction

Semantic Textual Similarity (STS) deals with determining the relations between sentences and documents. It is a regression task that inputs two texts and outputs a real value representing their meaning similarity. Because of its various applications, it is one of the essential tasks in Natural Language Processing (NLP). It is used in intent recognition, text summarization, question answering, sentiment analysis, and many other applications.

STS relies on how well an algorithm transforms sentences into vectors. In this work, different approaches for computing vector representation are explained. Some of the simplest embedding algorithms, such as Word2Vec or Sent2Vec, produce static word embeddings. They create sentence embeddings by combining the embeddings of individual words. Because of it, important information, for example, the word order, is lost.

Sentence embeddings generated by sequence models depend on the word order. They combine words more intelligently but suffer from problems like vanishing gradient or hardware under-utilization. The modern state-of-the-art algorithms use the transformer architecture. It solves the latter issues and also creates more sophisticated embeddings. This work aims to explain how transformers work and evaluate their performance on different datasets. These datasets include general sentences, sentences containing mistakes, typographical errors, ambiguity.

In order to encourage the use of a standard evaluation system of different algorithms on the STS task, the *STS Benchmark (STS-B)* dataset was proposed [1]. It consists of manually labeled pairs of sentences divided into training, validation, and test sets.

This work evaluates general models' performance on the STS task using the STS-B. It includes a comparison of predicted and referenced similarity scores using Mean Absolute Error (MAE) and Pearson correlation coefficient. MAE is calculated as an average of differences between referenced and predicted scores for given sentence pairs.

In later sections, the transformer architecture is evaluated on the STS-B dataset, augmented with grammatical and typographical errors. It is also tested on several complicated sentences containing lexical ambiguity to understand transformer behavior better.

This work is organized in the following way. Section II focuses on methodology. In subsection II-A, sentence embeddings are explained. Different embeddings strategies are shown in II-C and II-D. Modern state-of-the-art architectures are discussed in II-E. In II-F, it is explained how transformers can be used in real applications that deal with semantic similarity. In section III-A, transformer-based model sRoBERTa and probabilistic model Sent2Vec are compared on the STS-B dataset. Then, in III-B some of the biggest challenges of NLP are discussed. In subsections III-C and III-D, it is shown how transformers deal with some of these challenges. Finally, section IV shows how transformers can be fine-tuned on a specific domain.

II. Methods

A. Sentence embeddings

In order to process a text, any algorithm must transform it into some numerical form. One way of doing this is by creating a *one-hot* vector for each word. It contains 1 in the place of the index of a word in the dictionary and 0 everywhere else. Sentence embeddings are created by adding all individual vectors. This approach is simple, but as we can imagine, each vector would have a size equal to the vocabulary size and contain many zeros. This approach is ineffective, but its most crucial disadvantage is that the dot product between any two vectors is 0, so they are all perpendicular. It means that based on this approach, the relationship of words *"good"* and *"great"* is not any different from the relationship of *"good"* and *"the"*.

A better approach is to use learned vector representations called *embeddings*. These are dense vectors that capture the meaning of words. N-dimensional vector can represent N unique features. Words with similar meanings are close to each other in the vector space. Moreover, these representations can also capture different relationships between words, like comparatives-superlatives or singular-plural [2]. It is also possible to perform simple algebraic operations on embeddings. A famous example shows that *"king"* $-$ *"man"*

+ *"woman"* gives a vector that is very close to the vector representation of *"queen"* [3].

B. Similarity Metrics

Euclidean distance is an intuitive similarity metric. For two embeddings $\mathbf{u}$ and $\mathbf{v}$, it is defined as follows:

$$d(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{i=1}^{n}(u_i - v_i)^2} \tag{1}$$

The more similar $\mathbf{u}$ and $\mathbf{v}$ are, the smaller the distance. The main drawback of this metric is its independence on the vector direction, which can be crucial for semantics. Two embeddings, for example, of words *"good"* and *"great"* could have similar directions, but their magnitudes can be different because one word can be more frequent in a text than the other.

The *dot product* of $\mathbf{u}$ and $\mathbf{v}$ is defined as

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{n} u_i v_i \tag{2}$$

The dot product is a metric that depends on the direction. More similar elements are oriented similarly. The metric has a drawback - it is not normalized. Larger vectors will tend to have higher dot products even if they are not similar. If we normalize the dot product:

$$\cos(\alpha) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|} \tag{3}$$

we get the cosine of the angle α between $\mathbf{u}$ and $\mathbf{v}$. The *cosine similarity* metric depends only on vectors' direction. The higher the cosine of the angle between two vectors, the higher the similarity. This metric is used for the rest of this work.

Performance on the semantic textual similarity task depends significantly on how well an algorithm transforms sentences into vectors. Sentence embedding is very similar to word embedding, but its goal is to capture the meaning of the whole sentence.

The most straightforward way to create sentence embedding is by combining individual word embeddings. It works reasonably well, but some vital information may be lost. The modern state-of-the-art algorithms encode the entire sentence to capture information such as word order. The following sections explore how the sentence embedding algorithms work.

C. Static embeddings

One of the simplest unsupervised embedding algorithms is Word2Vec. It has two forms: Continuous Bag of Words (CBOW) and Skip-gram [4]. Skip-gram is a neural network with one hidden layer that takes a one-hot representation of an input word and tries to predict its context (the number of context words can be various and specified in the training phase). The algorithm attempts to map a set of inputs to a set of outputs during the training phase. However, this task's real goal is to learn the weights of the hidden layer. These weights are then used as embeddings of words for which the context was predicted. CBOW works similarly, but instead of taking a word and predicting its context, it takes a context and predicts the target word.

Word2Vec is created primarily for word embeddings. Sentence embeddings are produced by averaging embeddings of individual words. In the experiments part of this work, we explore Sent2Vec model, which can be viewed as an extension of Word2Vec, which includes *n-grams* (sequences of n words) encoding [5].

Word2Vec and Sent2Vec learn one static embedding for each word in the vocabulary. It is unable to capture multiple meanings of words. For example word *"back"* is encoded in the same way in both sentences: *"He is lying on his back"* and *"I will be back"* Its other disadvantage is the loss of word order in a sentence, so sentences *"Only she told him that"* and *"She told him only that"* will have identical embeddings.

D. Sequence models

Models based on a *Recurrent Neural Network (RNN)* proved to be accurate and efficient embedding algorithms. An RNN is a neural network in a loop. It is particularly suitable for processing sequential data such as sentences. It consists of the encoder and decoder parts. The encoder processes a sentence sequentially, word by word, creating a sentence embedding. It takes the previous step's output as its input, processes it, and propagates it further. The whole sentence is encoded and provided to the decoder at the last step. Because of the sequential nature, the final encoding depends on the word order.

The form of the decoder depends on the application. In case of the STS task, we are more interested in the encoder part of architecture because it generates embeddings. The details of the decoder part of RNNs are not explained.

In a classical RNN, sentences are processed sequentially from left to right. Sometimes it is worth knowing the context of a word from both sides. For this purpose, a *Bidirectional Recurrent Neural Network (BRNN)* was introduced. A BRNN consists of two independent RNNs. One processes the original sentence, and the other processes it in reverse. They provide backward and forward information (context) at each time step. The model's final output is the combination of both RNNs' outputs.

Despite being a powerful model, RNN has several drawbacks:

1) **Vanishing gradient**. During the training phase, gradients are propagated through neurons and through time. Many small numbers are multiplied, so the resulting gradient decreases quickly, approaching zero. Because of that, weights take a long time to update. It makes an RNN very difficult to train. Some RNN-based models like Long Short Term Memory (LSTM) help with this problem but do not solve it completely [8] [9].
2) **Hardware under-utilization**. Due to sequential computation, it is challenging to process RNNs on GPU or TPU. Hardware capacity is not used efficiently, making training and inference phases slower [10].

3) **Difficulty capturing long-term dependencies**. The encoded text is transferred to the decoder only by the intermediate state [7]. It can be insufficient for long texts because it is problematic to capture and learn all the relevant features using just one vector. To overcome this problem attention mechanism was introduced. It takes outputs from all hidden states and provides their weighted sum directly to the decoder. Weights are calculated so that the more important a word to a text, the more weight it has. These weights are learned automatically during the training phase.

E. Transformers

The *transformer* architecture revolutionized NLP. It was introduced in [11]. The paper's name, *"Attention is all you need,"* describes precisely the main idea of the architecture. The study proved that the attention mechanism in an RNN does a better job at encoding than an RNN itself does. The researchers introduced the transformer architecture with the *self-attention* mechanism, where an RNN was removed from both the encoder and decoder parts.

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based language model. Since it generates language representations, it consists of only the encoder part of the transformer. It is a self-supervised approach trained on a large amount of data, for instance, the entire Wikipedia. BERT uses two training strategies:

1) **Masked Language modeling** is a technique that masks randomly chosen words in a sentence and then trains a model to predict them. It makes BERT truly bidirectional, taking into account previous and following context simultaneously.
2) The **Next Sentence Prediction** technique is a classification layer of BERT, where during training, it gets two sentences and tries to predict if one goes after the other. It helps BERT to understand the relationship between pieces of text.

BERT is trained to perform general language understanding tasks. Nevertheless, it can also be fine-tuned on smaller task-specific datasets. For a regression task like STS, a regression layer can be added on top of the transformer's output and then trained using a labeled dataset. Similarly, a classifier on top for a classification task could be added. Simple datasets with only a couple of thousand examples can be enough for a good performance [13].

BERT processes sentences in parallel, which solves the vanishing gradient and hardware under-utilization problems. It, therefore, can be trained on larger amounts of data which improves its general performance.

In the experiments part of this work, a BERT derivate *Robustly Optimized BERT Pre-training Approach (RoBERTa)* is used. The research proved that the Next Sentence Prediction phase does not significantly improve the overhaul performance [14]. So in RoBERTa, this phase was removed entirely. The study also showed that BERT masks sentences during the preprocessing, implying that the number of sentence variations is bounded. RoBERTa masks sentences during the training phase solving this issue. The overhaul performance of RoBERTa on STS is similar to BERT's, but it is trained on ten times more data.

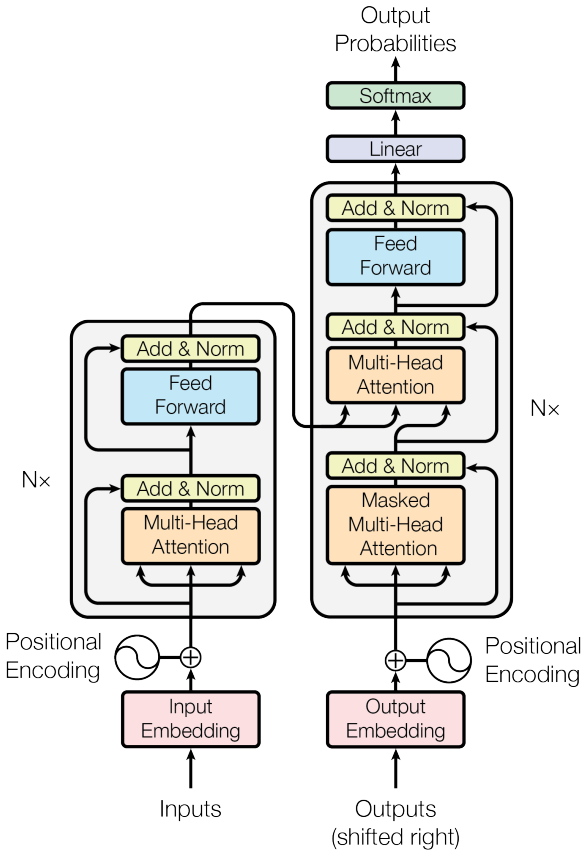

Fig. 1: The architecture of the transformer [12]

F. Sentence transformers

One of the main disadvantages of BERT in sentence-pair regression tasks is its computational complexity. It uses a *cross-encoder* architecture that inputs two sentences and outputs a number (a similarity score). Finding the most similar pair in a set of 10,000 sentences takes 49,995,000 inference computations or approximately 65 hours on a modern GPU [21]. It makes it almost impossible to use classical BERT for such tasks as STS.

One way of dealing with this problem is to precompute sentence embeddings and compare them using cosine similarity. The issue is that cross encoders do not compute embeddings for individual sentences. They could be retrieved by averaging the outputs, but it would produce inaccurate results.

The *bi-encoder* architecture overcomes this problem. It aims to map all the sentences to a vector space where embeddings of similar pieces of text are close to each other. It passes two sentences in a *Siamese* fashion. It means that sentences are processed by two identical independent sub-networks. The model fine-tunes pre-trained BERT to output embeddings corresponding to the reference using a labeled dataset (consisting of pairs of sentences with respective

scores). This approach allows the model to precompute embeddings in advance.

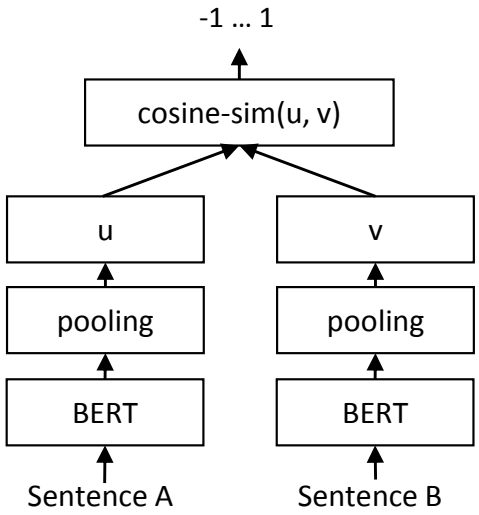

Fig. 2: The architecture of the bi-encoder [21]

Bi-encoders were proposed in [21]. They modified several popular models like BERT and RoBERTa, calling their implementation *Sentence-Transformers*. This approach is more computationally efficient but usually achieves lower performance and requires more training data than cross-encoders [21]. Nevertheless, it allows models like BERT and RoBERTA to be used for tasks requiring semantic textual similarity.

III. Results

The NLP field progresses quickly, so many different algorithms and approaches regarding semantic textual similarity can be considered. In the experiment part of this work, the focus is on the transformer architecture. First, transformer-based model sRoBERTa and Sent2Vec algorithm are compared on the STS-B dataset. Then sRoBERTa's performance is evaluated on more challenging examples to understand its behavior better.

Sent2vec was trained on English tweets (*twitter_unigrams* dataset), RoBERTa was pre-trained on the union of five datasets: *BookCorpus*, *English Wikipedia*, *CC-News*, *OpenWebText*, and *Stories* [15] and then fine-tuned on different datasets containing over 1 billion sentence pairs in total [16]. The fine-tuning is performed by providing a bi-encoder a set of labeled pairs of sentences.

The evaluation uses MAE and Pearson correlation coefficient. MAE is a simple measure of errors between predicted and intended values. Pearson correlation coefficient reflects the linear relationship between variables, which are predicted and referenced similarities in our case.

A. STS Benchmark

Sent2Vec and a transformer-based architecture such as RoBERTa require different amounts of data and training conditions to perform optimally. Since training takes a lot of time and resources, the comparison includes publicly available pre-trained models.

First, the algorithms are evaluated on the STS-B dataset, on which none of the models were pre-trained. Then sRoBERTa is fine-tuned on the STS-B training set to test if it increases the performance. The dataset contains 5749 training, 1500 validation, and 1379 test samples. Each sample consists of a pair of sentences and a manually assigned score representing their similarity from 0 to 5. A label 0 means that two sentences are entirely different, while 5 means complete equivalence.

model	fine-tuning	accuracy	Pearson
Sent2Vec	-	0.82	0.74
sRoBERTa	-	0.86	0.84
	STS-B	0.89	0.89

TABLE I: Evaluation of models on the STS-B testing set. The accuracy is calculated as $1 - \frac{1}{n}\sum_{i=0}^{n}|s_i - \hat{s}_i|$, where $\hat{s}_i$ is the prediction and s_i is the true similarity score, n is the number of compared sentences.

In table I, we can see that pre-trained sRoBERTa performs better than Sent2Vec on this dataset. Fine-tuning further improves the accuracy by 3%.

This comparison of the algorithms is somewhat shallow because models are trained on different data. Also, it is not easy to see how models behave in non-general cases. The focus of the following experiments is to explore more the behavior of transformers and their advantage over other models rather than comparing their general performance.

B. Challenges in NLP

Human language complexity presents many challenges for NLP. Some sentences can be said in many different ways. The others are hard to understand without context, even for a human. Let us consider an example. The first sentence is *"A car leaves its shed,"* the second is *"A tree shed its leaves."* These sentences consist of very similar words, but their meaning is entirely different. A good model will have very different embeddings for these two sentences, returning a low similarity score. Consider another example *"I enjoy eating apples"* and *"I found apples very tasty"* In this case, the texts are different, but it is evident that the second sentence is a rephrasing of the first one. A reasonable similarity score will be high.

Some texts can contain grammatical errors, slang, or sarcasm. These features are often tricky for the NLP system to understand [17]. The next sections explore some of these problems and evaluate a transformer-based model on some challenging examples.

C. Errors in text

Misspelled or grammatically incorrect words and sentences are not uncommon in a human language. Some of the typical grammatical mistakes in the English language are misspelled words. For instance, *"acquire"* is frequently wrongly spelled

as *"aquire."* Other grammatical errors include mixing up similar words like *"affect"* and *"effect"* and mistakes in articles (lack or misuse) [18], [19]. Other sources of errors in texts are typographical errors made during typing. Wrongly typed letters are often close to each other on the keyboard.

Many misspellings can be autocorrected, but it has limitations. Some sentences can lose their meaning if corrected the wrong way. For example, the sentence *"I have seven gosd coins"* is sometimes corrected as *"I have seven good coins."*[1]. In this case, *"gold"* is more appropriate, but an algorithm can select the most frequent matching word.

Some mistakes can be atomic typos. These typographical errors result in incorrectly spelled words that differ from the intended ones. It is often tough, if not impossible, to correct this type of mistake, even for a human. [20]. For example, when the word *"now"* is typed as *"not"* sentence *"The game is now available for the iPad"* becomes *"The game is not available for the iPad,"* which radically changes the meaning[2].

Because of the abovementioned reasons, it is worth evaluating the transformer's reaction to erroneous sentences that were not autocorrected. For this task, a dataset was created. It is derived from the STS-B. Each sentence was processed, and if it contained some typically misspelled word, it was replaced with it. With some probability, a typographical mistake was added to a target word, where characters were substituted by a keyboard distance.

model	fine-tuning	accuracy	Pearson
sRoBERTa	-	0.76	0.48
	STS-B	0.82	0.69
	STS-B with errors	0.85	0.78

TABLE II: Evaluation of differently tuned sRoBERTa on the STS-B testing set containing grammatical errors and typos. The accuracy is calculated as $1 - \frac{1}{n}\sum_{i=0}^{n}|s_i - \hat{s}_i|$, where $\hat{s}_i$ is the prediction and s_i is the true similarity score, n is the number of compared sentences.

In table II, the results are shown. We can see that the performance of sRoBERTa decreased significantly. It shows that transformers cannot deal with misspelled words very well. It is not surprising considering that this model does not consider characters n-grams.

One of the advantages of transformers is that they can be fine-tuned for a specific task. The fine-tuning is performed using a specific dataset. This dataset contains the STS-B train split augmented with mistakes. In table II, the results of evaluation on the STS-B set with errors are shown. We can see that fine-tuning with augmented data can improve the performance on a set with erroneous sentences, but it is still far from ideal.

[1]This example was evaluated in Notes application on MacOS

[2]example is taken from http://timcallan.com/blog/2011/04/23/the-now-vs-not-typo/

D. Ambiguity

Embeddings can be *static* or *dynamic* (*contextualized*). Some models, like Word2Vec and Sent2Vec, generate static embeddings and therefore cannot capture different meanings of words. Transformers are more powerful encoders. Because of the self-attention mechanism, word embeddings always depend on their context.

The meaning of some words cannot be identified without their surrounding context. For example, the word *"bank"* is ambiguous, so that it can have different interpretations depending on the surrounding words.

#	First sentence	Second sentence	Similarity	Sent2Vec	sRoBERTa
1	How are you	How old are you	low	0.68	0.26
2	I went to the river bank	I went to bank to make a deposit	low	0.62	0.59
3	She is going to the park	She is going to park her car	low	0.81	0.56
4	Put the stress on the second syllable	The examination put a lot of stress on him	low	0.53	0.21
5	The house is looking really run down	The house is in a really bad condition	high	0.43	0.89
6	We ran out of milk	We didn't have any more milk	high	0.61	0.94
7	The man took a bow to shoot	The man took a bow to the king	low	0.68	0.54
8	I enjoy eating apples	I found apples very tasty	high	0.58	0.80

TABLE III: Sent2Vec and sRoBERTa's prediction of semantic similarity of challenging sentences. Some of the examples are taken from https://monkeylearn.com/blog/natural-language-processing-challenges/

Transformers learn embeddings of words in their lowest layers and then correct them with the attention mechanism. A small dataset can evaluate the transformers' performance on challenging sentences. In table III, some pairs of sentences are shown. sRoBERTa predicts a score corresponding to sentence similarity. For reference, Sent2Vec results are also presented. Sent2Vec returns chaotic scores and does not perform well on this task because of the reasons mentioned earlier. It is worth noticing that it does not predict well even for such simple pair as the first one. It happens because the algorithm creates sentence embedding by averaging individual words (in this case, they are all the same except the word *"old"*). In the transformer's case, we can see that some of the predicted values are not perfect (the second, third, and seventh scores could be lower). Nevertheless, considering the task's difficulty,

the model performs reasonably well, and in most cases, prediction corresponds to the similarity label. If the threshold for the sentence similarity is set to 70, the classification performance of these sentences is 100%.

IV. Fine-tuning

As was previously discussed, the meaning of some words is context-dependent. The definition of commonly used words can be domain-specific. For example, chemists most likely think of a substance when using the word *"element,"* while mathematicians can understand it as a member of a set. Some words are used only in a specific domain. They are unusual to everyday speech.

Sentence transformers can be fine-tuned by providing them a set of tuples of the form *(sentence1, sentence2, score)*. Then a model can be trained in a supervised fashion to create more suitable sentence embeddings.

A new dataset was created to evaluate how transformers behave on different domains. The dataset contains some sentences and facts from the book *"Harry Potter"* by J.K. Rowling. It consists of many uncommon words, like *"Wingardium leviosa"* or *"Avada Kedavra."* The meaning of some words differs from the common usage. For example, the word *"house"* in this case refers to a *"group"* or a *"community."* Some sequences of words like *"He-Who-Must-Not-Be-Named,"* which specify one of the characters, have a unique meaning in this specific domain.

The dataset consists of 618 samples, of which 415 samples are used for training, 100 for validation, and 94 for testing. The dataset was created in the following way:

1) A small number of sentences (108 in total) was created by hand. These sentences contain general facts from the *"Harry Potter"* books. An example of such a sentence is *"Harry Potter can be recognized by his cursed scar."*
2) A separate list of synonyms was made. The list contains other names of important entities. For example, *"Harry Potter"* is also known as *"The Boy Who Lived."*
3) Equal pairs are generated using synonyms. The first sentence is the original one. The second is equal to the first, with all the entities replaced with their respective synonyms. For example: *"Harry Potter can be recognized by his cursed scar"* and *"The Boy Who Lived can be recognized by his cursed scar."* The sentences are considered equal in this experiment, so their similarity score is 1.
4) Different examples (with similarities different from one) can be created by randomly choosing two sentences from a list of facts and manually labeling them. In this work, sentences were labeled using the following rules:
   a) 1.0 score is assigned to a rephrased sentence (as was discussed above)
   b) 0.75 score is for sentences that are not the same but share a meaning. Example: *"Harry Potter can be recognized by his cursed scar"* and *"Harry Potter had a lightning-shaped scar."*
   c) 0.5 score for sentences that share a topic. For example: *"Capturing the Golden Snitch ends the game immediately," "The Golden Snitch is worth 150 points."*
   d) 0.25 score for related sentences (sharing entities or other related information). Example: *"Slytherin house is represented by a serpent"* and *"Harry Potter would fit well in Slytherin house"*
   e) 0.0 for all the others.

It is worth pointing out that these rules and values apply to this particular task and can be modified to suit a particular application.

model	fine-tuning	accuracy	Pearson
sRoBERTa	-	0.663	0.751
	Harry Potter	0.907	0.975

TABLE IV: Comparison of pre-trained and fine-tuned on a domain sRoBERTA on Harry Potter testing set. The accuracy is calculated as $1-\frac{1}{n}\sum_{i=0}^{n}|s_i-\hat{s}_i|$, where $\hat{s}_i$ is the prediction and s_i is the true similarity score, n is the number of compared sentences.

In table IV the resulting performance is shown. We can see that overhaul performance without additional tuning is already reasonable. Nevertheless, just a small amount of training examples can improve the performance even more.

The fine-tuning takes around 22 minutes per epoch on the M1 chip. It takes around 60 seconds to encode and compare the sentences from the test split. It is a long time. Nevertheless, sentence embeddings can be precomputed. If a set is fixed, it can be encoded in advance to further compare to a query (for example, user input). In this case, only a query is encoded at runtime. So the inference phase lasts 0.4 seconds on M1 chip or 0.05 seconds on a modern GPU.

V. Conclusion

Semantic textual similarity is an essential task in NLP. The modern state-of-the-art techniques dealing with STS are based on the transformer architecture. It provides embeddings that allow a model to understand different meanings of words and sentences. It is particularly interesting for languages that use ambiguous words.

Because of their computational complexity, classical transformers like BERT are very inefficient in real semantic textual similarity applications. The bi-encoder architecture solves the issue. It allows precomputing sentence embeddings, making the inference phase computationally efficient.

In later sections, transformers were tested on the STS Benchmark dataset. Then some of the challenges of NLP were discussed. Modern transformer-based architectures can deal with some of these challenges, but they have limitations. For example, transformers can deal with long-term dependencies and ambiguities. However, it is challenging for them to perform well on sentences containing errors. They do not encode characters n-grams, so they do not process erroneous sentences adequately. The encoding can be improved by fine-tuning a model with an augmented dataset. However, the Pearson correlation coefficient is still 10% worse than the

original. The results are less correlated, indicating that the sentences' real meaning is most likely lost.

The last section shows how sentence-transformers can be fine-tuned on a specific domain language using a dataset based on *"Harry Potter"* books. It also shows how a dataset could be created for a particular domain. It can be concluded that by providing the model with just a few hundred training examples, the accuracy can improve by 20-25%. It is worth mentioning that based on the evaluation of a simpler dataset (table III-A), the fine-tuning does not show the same improvements. It happens because the pre-trained model was already trained on a large amount of data. So the STS-B dataset used for the fine-tuning does not bring much new information. On the other hand, a created dataset based on *"Harry Potter"* books introduced new facts to the model. So the resulting performance is improved significantly.

REFERENCES

[1] Cer, Daniel, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. *SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation*. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 1–14. Vancouver, Canada: Association for Computational Linguistics. https://aclanthology.org/S17-2001/

[2] Liu, Shusen, Peer-Timo Bremer, Jayaraman J. Thiagarajan, Vivek Srikumar, Bei Wang, Yarden Livnat, and Valerio Pascucci. 2018. 'Visual Exploration of Semantic Relationships in Neural Word Embeddings'. IEEE Transactions on Visualization and Computer Graphics 24 (1): 553–62. https://doi.org/10.1109/TVCG.2017.2745141.

[3] Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig. 2013. 'Linguistic Regularities in Continuous Space Word Representations'. In Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 746–51. Atlanta, Georgia: Association for Computational Linguistics. https://aclanthology.org/N13-1090.

[4] Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. 'Efficient Estimation of Word Representations in Vector Space'. ArXiv:1301.3781 [Cs], September. http://arxiv.org/abs/1301.3781.

[5] Pagliardini, Matteo, Prakhar Gupta, and Martin Jaggi. 2018. 'Unsupervised Learning of Sentence Embeddings Using Compositional N-Gram Features'. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), 528–40. https://doi.org/10.18653/v1/N18-1049.

[6] Jurafsky, Dan, and James H. Martin. 2009. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Prentice Hall.

[7] Bengio, Y., P. Simard, and P. Frasconi. 1994. 'Learning Long-Term Dependencies with Gradient Descent Is Difficult'. IEEE Transactions on Neural Networks 5 (2): 157–66. https://doi.org/10.1109/72.279181.

[8] Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. 2013. 'On the Difficulty of Training Recurrent Neural Networks'. ArXiv:1211.5063 [Cs], February. http://arxiv.org/abs/1211.5063.

[9] Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. 'Long Short-Term Memory'. Neural Computation 9 (December): 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735.

[10] Peng, Lu, Wentao Shi, Jian Zhang, and Samuel Irving. 2019. 'Exploiting Model-Level Parallelism in Recurrent Neural Network Accelerators'. In 2019 IEEE 13th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip (MCSoC), 241–48. https://doi.org/10.1109/MCSoC.2019.00042.

[11] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. 'Attention Is All You Need'. ArXiv:1706.03762 [Cs], December. http://arxiv.org/abs/1706.03762.

[12] Rogers, Anna, Olga Kovaleva, and Anna Rumshisky. 2020. 'A Primer in BERTology: What We Know about How BERT Works', February. https://arxiv.org/abs/2002.12327v3.

[13] Sun, Chi, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. 'How to Fine-Tune BERT for Text Classification?', May. https://arxiv.org/abs/1905.05583v3.

[14] Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. 'RoBERTa: A Robustly Optimized BERT Pretraining Approach'. ArXiv:1907.11692 [Cs], July. http://arxiv.org/abs/1907.11692.

[15] 'Roberta-Large · Hugging Face'. n.d. Accessed 23 December 2021. https://huggingface.co/roberta-large.

[16] 'Sentence-Transformers/All-Roberta-Large-v1 · Hugging Face'. n.d. Accessed 23 December 2021. https://huggingface.co/sentence-transformers/all-roberta-large-v1.

[17] Kaddari, Zakaria, Youssef Mellah, Jamal Berrich, Mohammed Belkasmi, and Bouchentouf Toumi. 2021. 'Natural Language Processing: Challenges and Future Directions'. In , 236–46. https://doi.org/10.1007/978-3-030-53970-2_22.

[18] Fengjie, Li. 2016. 'Grammatical Mistakes in College English Writing: Problem Analysis, Re'Roberta-Base · Hugging Face'. n.d. Accessed 23 December 2021. https://huggingface.co/roberta-base. asons and Solutions'. International Journal of Applied Linguistics and Translation 2 (3): 20. http://www.sciencepublishinggroup.com/journal/paperinfo?journalid=340&doi=10.11648/j.ijalt.20160203.11

[19] Lunsford, Andrea A., and Karen J. Lunsford. 2008. '"Mistakes Are a Fact of Life": A National Comparative Study'. College Composition and Communication 59 (4): 781–806. https://www.jstor.org/stable/20457033

[20] Callan, Tim. n.d. 'The Now vs. Not Typo — Tim Callan on Marketing and Technology'. Accessed 14 December 2021. http://timcallan.com/blog/2011/04/23/the-now-vs-not-typo/.

[21] Reimers, Nils, and Iryna Gurevych. 2019. 'Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks'. ArXiv:1908.10084 [Cs], August. http://arxiv.org/abs/1908.10084.