Semantic Similarity Matching Using Contextualized Representations

Farhood Farahnak[†], Elham Mohammadi^{‡,*}, MohammadReza Davari^{‡,*}, Leila Kosseim[†]

[†] Dept. Computer Science and Software Engineering Concordia University, Montreal, Canada

[‡] Coveo NLP Labs, Montreal, Canada

Abstract

Different approaches to address semantic similarity matching generally fall into one of the two categories of interaction-based and representation-based models. While each approach offers its own benefits and can be used in certain scenarios, using a transformer-based model with a completely interaction-based approach may not be practical in many real-life use cases. In this work, we compare the performance and inference time of interaction-based and representation-based models using contextualized representations. We also propose a novel approach which is based on the late interaction of textual representations, thus benefiting from the advantages of both model types.

Keywords: Semantic similarity, Late interaction, Contextualized representations, Transformers, BERT

1. Introduction

Semantic Similarity Matching (SSM) between texts is one of many sub-tasks in Natural Language Understanding (NLU) with a wide range of downstream applications such as information retrieval, question answering, and paraphrase detection. Traditionally, models which were based on word frequency or word embedding representations were used for similarity evaluation [1–3]. These models can be fast and effective in several cases; however, they cannot capture the semantic similarity between two pieces of texts when the same concept is referred to using two completely different wordings.

Recently, transformer-based language models [4–6] have achieved state-of-the-art performance in many tasks [6–8] including textual sequence matching [9]. However, these models are computationally expensive which makes it hard to apply them in a real-life scenario. In many industrial use cases, models are required to provide a response in a few milliseconds, while being limited in accessing efficient GPUs. The combination of these two restrictions makes it vital to develop models that are efficient and scalable as well as effective given certain performance metrics.

Previous works on SSM either focus on scalability or performance of the system. These approaches generally fall into one of the two following categories: Those that are based on the interaction between two textual sequences [10–13] and those that are based on static representations, paving the way for the pre-computation and reusability of those representations [14–16].

Motivated by the need for scalable systems that meet certain performance criteria for real-world use cases, in this work, we compare the performance of different models, ranging from fully interaction-based to fully representation-based, in addressing the SSM task.

2. Models

Figure 1 shows the general architecture of the 3 models that were experimented with. In all of these models, a transformer-based encoder is used to produce the representation of the input texts. The encoder creates a contextualized embedding for each token. A *[cls]* token is added to the beginning of each text and its corresponding embedding is used as a

[†]{farhood.farahnak, leila.kosseim}@concordia.ca [‡]{emohammadi, rdavari}@coveo.com

*Equal contribution

summary representation of the input text (in previous work, the resulting representation of the $[cls]$ token has been used for different tasks such as classification).

In all of the 3 models, a single feed-forward layer with Sigmoid activation functions is used as the similarity module. However, in a ranking task, one can simply replace the classification layer with cosine similarity or other ranking schemes and use the same strategies for the task.

Figure 1. Model architectures; in the attention module of the Merge model, dotted lines are used for queries and solid lines for keys and values.

Fully Interaction Based (FI). In this scenario, we followed the strategy used in previous work [6], where both pieces of text are concatenated together and passed to the encoder as a single input sequence. The embedded representation of the $[cls]$ token is given to the classifier module to determine whether or not these two texts are similar.

Fully Representation Based (FR). In this model, the encoder module separately encodes the two pieces of text resulting in two distinct representations for each text. Then the summary representation of each sequence (the $[cls]$ token) is extracted from their corresponding encoded representation. These summary representations are then concatenated and passed to the classification layer.

Late Interaction Based (LI). In the FR model, we used the embedding vector of the $[cls]$ token as a representation for each textual sequence, but each of these representations has no information about the other sequence. In other words, the summary representation for a piece of text is generated in a static manner, independent of the other sequence with which a similarity score will be computed. In order to remedy this issue, we propose a novel architecture that allows late interactions between the sequence representations. The LI model creates a new representation for each sequence by taking into account the information that is present in both sequences. In this model, each text is passed to the encoder separately. In order to produce a single representation for each text, we use a multi-head attention mechanism, where the keys and values are token representations of that text and the query is the $[cls]$ token representation of the other text. The attention module calculates a weighted sum over all token representations for a given sequence, where the weights (attention scores)

Table 1. Dataset statistics.

Subset	# Samples	% of Positive
Train	324234	0.34
Validation	40000	0.46
Test	40000	0.46

Table 2. Model performance on the test set.

Model	Accuracy	F1	Train Time Per Epoch
FI	90.57	89.90	1052s $\pm$ 4
LI	85.54	84.92	1292s $\pm$ 7
FR	83.83	82.79	1299s $\pm$ 8
Baseline	77.55	75.81	105s $\pm$ 10

Table 3. Inference time on GPU.

Model	# Queries	Inference Time
FI	1	13.77s $\pm$ 0.23
	4	56.67s $\pm$ 0.96
LI	1	0.81s $\pm$ 0.02
	4	2.02s $\pm$ 0.02
FR	1	0.07s $\pm$ 0.00
	4	0.07s $\pm$ 0.00
Baseline	1	0.03s $\pm$ 0.00
	4	0.08s $\pm$ 0.00

are calculated using the dot product of the token representation of the first sequence and the $[cls]$ token representation of the second sequence. The $[cls]$ token representation of the second text holds its summary information and the dot product can capture the alignment between this summary representation and the token representations of the other text. As a result, tokens that are semantically more similar to the summary of the other text have greater weights in the attention module. Therefore, the new representations that are given by the attention module contain information on the relationship between the two sequences and how they interact with each other. We apply the multi-head attention module on both texts (each text treated one time as the query and one time as the keys and values) to get a new representation for each text. At the end, these two representations are concatenated and passed to the classifier which labels them as similar or dissimilar.

Baseline. In order to better study the performance of our models, we also utilize a simple feed-forward neural network following previous work [17] as a baseline model. For this model, the average of non-contextual word embeddings for the tokens in each text is calculated as a summary representation. The two summary representations are then concatenated and passed as input to the feed-forward model. *Baseline* in Table 2 refers to this model.

3. Experiments and Results

3.1. Implementation

We use DistilBERT [18] as our pre-trained language model from the Transformers library [19]. The DistilBERT model has 6 encoder layers and is trained by leveraging knowledge distillation, resulting in a faster and smaller model than the original BERT [6]. The models used in our experiments are implemented using the PyTorch library [20]¹. In order to train the models, the maximum sequence size is set to 512 tokens and batch size is set to 16. Models are trained for 8 epochs with a learning rate of 10^{-5} for the Encoder module and 10^{-4} for all other modules. For our baseline model, we use the 300d version of GloVe (pretrained on 6B tokens) [21]. The training and evaluation of the models are performed on Amazon EC2 ² **p3.2xlarge** machines. For the evaluation of inference-time performance on CPU, experiments are performed on an **i3.2xlarge** instance.

3.2. Dataset and Results

The dataset used in this work is the Quora Question Pairs³ dataset (QQP), which contains 404,290 question pairs. In order to train and evaluate our models, we divide the original dataset into training, validation, and test sets. Table 1 presents a summary of each subset.

¹The code and models are publicly available at <https://github.com/farhoodf/SSCORE>

²<https://aws.amazon.com/ec2/instance-types/>

³<https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs>

Table 2 shows the performance of each model on the test set. The FI model, which is the main approach for this task [6], achieves the highest performance while the FR model scores the lowest among the transformer-based models in our evaluation. The LI model achieves an approximate 2% improvement in terms of F1 score compared to the FR model. Table 2 also shows the average time required to train each model for one epoch. As shown in the table, the FI model is faster than the others in terms of training time. This is because in the case of LI and FR models, we pass each piece of text in a question answer pair separately to the model, which can be seen as a double batch size enforced on these models compared to the FI model. However, the length of sequences in the case of the FI model is twice the sequence length in the other 2 models.

3.3. Inference Time

During the inference time on the test set, we need to compare each new query with all the samples that we already have in our dataset. In order to have a better understanding of the inference time of our models in a real-life scenario, we conduct a few experiments in which a single query is compared with all the samples in our test set (i.e 40k samples). For each model, we utilize the following optimization techniques to reduce the inference time:

FI. For this model, all the samples in the test set are pre-tokenized. Note that since this model is fully interactive, we cannot precompute any kinds of vector representations for the test cases.

LI. All the test samples are fed to the encoder module and their representations are pre-computed and stored in memory. During the inference time, each query is first passed to the encoder, after which the representation of the query along with the representations of all test samples are passed to the multi-head attention module. In the end, the output of the attention module is passed to the similarity module.

FR. Similarly to the LI model, first, all the test samples are encoded. But in this case, instead of storing the output vectors for all the tokens, only the summary representations (that is the $[cls]$ token representations) are stored. During inference, the summary representation of the query alongside the summary representations of all test samples are passed to the similarity module.

In all experiments, a batch size of 500 for the test samples is used as it results in the optimum running time for all models. As Table 3 shows, using the FI model is not feasible in a real-life scenario as, in many use cases, a response should be provided in less than a second. In the case of the LI model, although it can handle a single query in less than a second, on a single GPU, it can not handle multiple queries under the one second threshold (see Table 3, processing 4 queries takes approximately 2 seconds). As a result, if we want to utilize this model in an industry application, we should either make use of multiple GPUs (resulting in one GPU used for each incoming query stream) or reduce the number of test cases (with which a similarity score needs to be computed) to a manageable number (e.g. ≈ 1000) using approximation algorithms, as the former solution may not be favored due to increased expenses.

3.4. Inference Time on CPU

As there might be restrictions on the use of GPUs in a real-world scenario, we also performed a few experiments to measure and try to optimize the inference time on CPUs. Since the LI and FR models are too expensive to be used on CPUs, we only used the FR model for these experiments. In order to better understand the behaviour and the running time of the model, the inference stage is broken down into 2 separate steps: (1) creating the summary representation for the query and (2) comparing the query summary with all test summaries.

Table 4. Scalability evaluation of the encoder module of the FR model.

Threads	Quantization	Max QPS	Latency
16	No	32	18.86 $ms \pm 2.17$
16	yes	32	18.86 $ms \pm 2.74$
4	No	64	34.80 $ms \pm 5.31$
4	yes	128	23.44 $ms \pm 3.91$
1	No	128	85.73 $ms \pm 13.80$
1	yes	256	51.91 $ms \pm 9.91$

Table 5. Scalability Evaluation of the similarity module of the FR model.

Threads	Quantization	Max QPS	Latency
16	No	4	174.12 $ms \pm 26.63$
16	yes	4	191.93 $ms \pm 25.09$
4	No	8	313.32 $ms \pm 43.67$
4	yes	8	353.45 $ms \pm 52.96$
1	No	8	990.00 $ms \pm 20.24$
1	yes	8	1050.00 $ms \pm 50.73$

The Encoder Module. In this stage, we receive a new query and create its summary representation using the encoder module. As Table 4 shows, a higher number of CPU threads leads to a decrease in the latency; however, it also causes a decrease in the number of Queries Per Second (QPS) that a system can handle. Therefore, there is a trade-off between the latency and QPS. If a higher QPS is required, the number of threads needs to be reduced, resulting in the latency being sacrificed to handle a higher QPS. On the other hand, if a use case requires a lower number of QPS, then, a higher number of threads can be used to have a better latency. In these experiments, we also measure the effect of dynamic quantization introduced by PyTorch. As it can be seen in Table 4, quantization does not have a significant effect when using 16 threads. On the other hand, it has a notable impact when a single thread is used (a reduction of $\approx 34ms$) and can even lead to a higher number of QPS.

The Similarity Module. In these set of experiments, we measure the running time of the similarity module on CPUs with different settings. As Table 5 shows, using quantization is not beneficial in this stage. Also, as expected, compared to the encoder stage, the similarity stage is the main bottleneck for this task. As a result, if the model is expected to run on CPUs, the number of samples in the test set should be limited using an approximation algorithm.

4. Conclusion

In this work, we experimented with an interaction-based (FI) and a representation-based (FR) model using different setups to evaluate the similarity between two textual sequences. We also proposed a novel strategy based on late interactions (LI) using contextual language models, which can be considered a compromise between interaction-based and representation-based models.

The models were evaluated in terms of performance as well as the required inference time. We found that, although the FI model achieved the best performance in terms of accuracy and F1 score, it is not efficient enough to be utilized for retrieval tasks where each new query needs to be compared with several other samples (in the case of our dataset, 40k samples), even when GPU machines are available. On the other hand, although the FR model had the lowest performance among the transformer-based models in our evaluation, it is the most efficient in terms of inference time and can be used in situations where there is no access to GPUs or in use cases where the inference time is the most important factor in choosing a model. Finally, when GPU resources are available, the use of the LI model can result in better performance compared to the FR model, and more efficiency compared to the FI model, making it a feasible candidate in a real-life scenario.

References

- [1] M. Surdeanu, M. Ciaramita, and H. Zaragoza. “Learning to rank answers on large online QA collections”. In: *Proceedings of the 2008 Annual Meeting of the Association for Computational Linguistics: Human Language Technology Conference (ACL:HLT 2008)*. 2008.

- [2] M. Surdeanu, M. Ciaramita, and H. Zaragoza. “Learning to rank answers to non-factoid questions from web collections”. In: *Computational Linguistics* 37.2 (2011).
- [3] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger. “From word embeddings to document distances”. In: *International conference on machine learning*. 2015.
- [4] 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”. In: *arXiv preprint arXiv:1907.11692* (2019).
- [5] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. “ALBERT: A lite BERT for self-supervised learning of language representations”. In: *International Conference on Learning Representations (ICLR 2020)*. 2020.
- [6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (NAACL 2019)*. Association for Computational Linguistics, 2019.
- [7] F. Farahnak and L. Kosseim. “Using Conditional Sentence Representation in Pointer Networks for Sentence Ordering”. In: *Proceedings of the 15th International Conference on Semantic Computing (ICSC 2021)*. IEEE. 2021.
- [8] M. Davari, L. Kosseim, and T. Bui. “TIMBERT: Toponym Identifier For The Medical Domain Based on BERT”. In: *Proceedings of the 28th International Conference on Computational Linguistics (COLING 2020)*. International Committee on Computational Linguistics, 2020.
- [9] S. Wang, Y. Lan, Y. Tay, J. Jiang, and J. Liu. “Multi-level Head-wise Match and Aggregation in Transformer for Textual Sequence Matching”. In: *arXiv preprint arXiv:2001.07234* (2020).
- [10] B. Hu, Z. Lu, H. Li, and Q. Chen. “Convolutional neural network architectures for matching natural language sentences”. In: *Advances in neural information processing systems*. 2014.
- [11] Y. Wu, W. Wu, C. Xing, M. Zhou, and Z. Li. “Sequential Matching Network: A New Architecture for Multi-turn Response Selection in Retrieval-Based Chatbots”. In: *Proc. of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017.
- [12] Q. Chen, Q. Hu, J. X. Huang, and L. He. “CA-RNN: using context-aligned recurrent neural networks for modeling sentence similarity”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.
- [13] M. Tan, C. Dos Santos, B. Xiang, and B. Zhou. “Improved representation learning for question answer matching”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2016.
- [14] J. Mueller and A. Thyagarajan. “Siamese recurrent architectures for learning sentence similarity”. In: *thirtieth AAAI conference on artificial intelligence*. 2016.
- [15] M. Seo, T. Kwiatkowski, A. Parikh, A. Farhadi, and H. Hajishirzi. “Phrase-Indexed Question Answering: A New Challenge for Scalable Document Comprehension”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018.
- [16] S. Wang, Y. Lan, Y. Tay, J. Jiang, and J. Liu. “Multi-level Head-wise Match and Aggregation in Transformer for Textual Sequence Matching”. In: *Proc. of the 2020 AAAI conference*. 2020.
- [17] L. Sharma, L. Graesser, N. Nangia, and U. Evci. “Natural language understanding with the quora question pairs dataset”. In: *arXiv preprint arXiv:1907.01041* (2019).
- [18] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *NeurIPS EMC2 Workshop*. 2019.
- [19] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP 2020)*. Association for Computational Linguistics, 2020.
- [20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019.
- [21] J. Pennington, R. Socher, and C. D. Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*. 2014.