UAICS at SemEval-2020 Task 4: Using a bidirectional transformer for Task A

Ciprian-Gabriel Cusmuliuc **Lucia-Georgiana Coca** **Adrian Iftene**
"Alexandru Ioan Cuza" University of Iasi, Faculty of Computer Science
cusmuliuc.ciprian.gabriel@info.uaic.ro
coca.lucia.georgiana@info.uaic.ro
adiftene@info.uaic.ro

Abstract

Commonsense Validation and Explanation has been a difficult task for machines since the dawn of computing. Although very trivial to humans it poses a high complexity for machines due to the necessity of inference over a pre-existing knowledge base. In order to try and solve this problem the SemEval 2020 Task 4 - "Commonsense Validation and Explanation (ComVE)" aims to evaluate systems capable of multiple stages of ComVE. The challenge includes 3 tasks (A, B and C), each with it's own requirements. Our team participated only in task A which required selecting the statement that made the least sense. We choose to use a bidirectional transformer in order to solve the challenge, this paper presents the details of our method, runs and result.

1 Introduction

Natural Language Understanding (NLU) is a topic that in recent years caught researchers' attention. Growing interest has led to the development of multiple areas of study, one such area being represented by "Commonsense Validation" which aims to evaluate a system's ability to infer commonsense knowledge from basic sentences.

The "SemEval 2020 Task 4 - Commonsense Validation and Explanation (ComVE)" (Wang et al., 2020) seeks to research in the aforementioned field by testing systems that are able to differentiate between statements that make sense from the ones that don't. The task is split in multiple tasks: task A implies selecting which statement is against commonsense between two very similar phrases, task B requires selecting between three reasons that explain why the statement from task A is incorrect and finally task C requires the machine to generate reasons why the statement from task A is against common sense.

Our team participated only in task A, in the official results our system ranked 17^{th} out of 45 submission with an accuracy of 89.1%, the best result being 97% and the worst 0%. The strategy was to use a pre-trained model, BERT (Devlin et al., 2018), in order to find out which sentence makes the most sense. This model has been trained and tested using different parameters in order to achieve the best result and it will be described with more details in section 2, "System Overview".

Similar work has been done by (Trinh and Le, 2018) where they trained a RNN on a very large text collection resulting in a 63.7% accuracy in the Winograd Schema Challenge (Levesque et al., 2012) while (Kocijan et al., 2019) using a BERT model was able to achieve overall accuracy of 72.5%; this performance increase led us to choose the latter model as we feel it best fits our purpose.

In this paper we will describe our system, results and conclusions. The structure is as follows: Section 2 will present the system designed and the experimental setup, Section 3 will talk about the result and in the end we will draw some conclusions.

Proceedings of the 14th International Workshop on Semantic Evaluation, pages 609–613
Barcelona, Spain (Online), December 12, 2020.

2 System overview

In this section we will present the architecture and inner workings of our solution, the system uses Python[1] (Anaconda[2]) combined with Torch[3] and Huggingface's Transformers[4] (that implements the BERT model) in order to achieve the task requirements.

We did not use external data sets, the only information used was contained in the BERT model that was in the Transformers library, more so no other external input sources besides the data sets provided by the organiser have been integrated.

The system output consisted of the sentence id followed by the index of the sentence against common sense (either 0 or 1).

2.1 Training data

The data set has been provided by the organiser (Wang et al., 2019) and contains the following: training set contained 10.000 sentence pairs, trial set 2.021 pairs, dev set 997 pairs and the test set 1.001. It must be noted that the dev and test data set (example in Table 2) did not have ordered ids as the other sets.

A training data example can be seen in Table 1, it is comprised of a sentence id, two phrases and a correct choice (answer).

id	Sentence 0	Sentence 1	answer
3	A mosquito stings me	I sting a mosquito	1

Table 1: Training data example.

id	Sentence 0	Sentence 1
1395	Everyone hates paying taxes	Nobody hates paying taxes

Table 2: Test data example.

2.2 Preprocessing

Before feeding the data to the model, we had to preprocess the text. We used the csv[5] library to read from the files provided by the organiser after which we would put the training data in a list that contained in order the id, sentence 1, sentence 2 and label. The data would then be sent to a tokenizer, we decided to use BertTokenizer[6] as this was the official API supported way and the model is very specific about the received input, after which the tokenized sentence would be padded to meet a certain length (we were forced by some model limitations to set a maximum phrase length, in our case 40, then pad the rest).

Example tokenizer input and output:

Input: "He poured orange juice on his cereal."
Output: [101, 2002, 8542, 4589, 10869, 2006, 2010, 20943,1012,102]

[1]https://www.python.org/
[2]https://www.anaconda.com/
[3]https://pytorch.org/
[4]https://huggingface.co/transformers
[5]https://docs.python.org/3/library/csv.html
[6]https://huggingface.co/transformers/model_doc/bert.html#berttokenizer

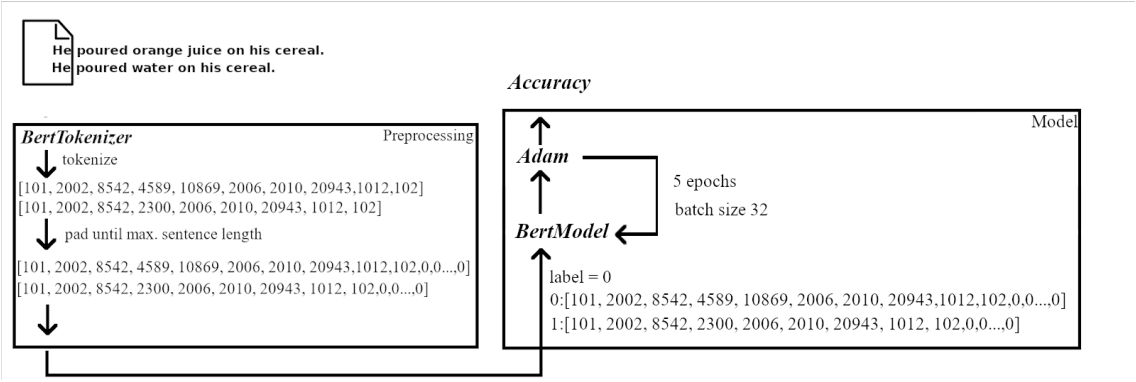

Figure 1: System diagram

After tokenization we loaded each individual field in a torch tensor[7] and inserted them in a TensorDataset[8] that contained: all the sentence ids, each individual tokenized sentences and the labels. A code snippet for this operation is the following:

```
all_sent1_id = torch.tensor([f.sent1_id for f in features], dtype=torch.long)
all_sent2_id = torch.tensor([f.sent2_id for f in features], dtype=torch.long)
all_label_id = torch.tensor([f.label_id for f in features], dtype=torch.long)
dataset = TensorDataset(all_sent1_id, all_sent2_id, all_label_id)
```

2.3 Model

In the designing of this model we took inspiration from a similar competition we participated described in (Coca et al., 2019) where a system that was capable of spotting check-worthy claims in English should be designed. It was there that we learned that a simple machine learning or a neural network trained on a data set was not enough for such a complex task and thus we chose a pre-trained model such as BERT, more so numerous papers emphasizing on the performance increases BERT brings over older systems such as ULMFiT (Howard and Ruder, 2018) have made us believe it might be a viable solution.

The system consists of a pre-trained model called "bert-large-uncased"[9], it is a bidirectional transformer that contains 24 layers, 1024 hidden layers, 16 heads and 340 million parameters. The training has been done on lower-cased english text.

We considered that since the model is already trained "world understanding" wouldn't be a problem thus only the training data set had been used. Also besides the preprocessing phase no other modifications on the data set have been made.

We used a combination of BertModel[10] and Adam[11] optimizer (Kingma and Ba, 2014) in order to get the best results. The hyperparameters are more or less standard, we tuned them empirically and arrived at the following best configuration: batch size 32, 5 epochs and the adam learning rate of 5e-5.

The pipeline of the algorithm implies preprocessing (tokenize and pad the sentence in order to satisfy the condition of the model) then we shuffle the data in order to avoid overfitting, we start the training with each epoch and we feed the data thought the network, backpropagate then simply update the learning rate and tell the optimizer to update the parameters. The model is standard but very efficient, leading to very good results, as it will be seen in Section 3.

Evaluation of the trained network is done with the trial dataset, using the saved model in the previous step we feed the data through the network and compute loss on our trial data.

In Figure 1 the system diagram is presented, illustrating the pipeline of the model.

[7] https://pytorch.org/docs/stable/tensors.html
[8] https://pytorch.org/docs/stable/data.html#torch.utils.data.TensorDataset
[9] https://huggingface.co/transformers/pretrained_models.html
[10] https://huggingface.co/transformers/model_doc/bert.html#bertmodel
[11] https://huggingface.co/transformers/main_classes/optimizer_schedules.html#adamw

2.4 Experimental setup

After designing the system we had to do experiments in order to establish what parameters to use, for this purpose we used the two data sets provided by the organisers, i.e.: train and trial. We did not split them, the strategy was very simple, train was used to train the model and trial was used to obtain performance measures.

Experiments have the following pipeline: change a hyperparameter, let it train for some time, obtain the accuracy, analyze and decide what to change further. The results of the model can be seen in the following chapter.

Before trying to test on a GPU the model was trained only with a 12 core CPU and 32 Gb of ram, setting which proved to be very inefficient as training time was very lengthy, taking up to 24 hours with 5 epochs and 48 hours with 10 epochs. We tried running the model in Google Colaboratory [12] and managed to train on GPU however due to time constraints we were not able to run as many experiments as we wanted or to fully optimize the model.

3 Results

In this section we will discuss the different results of the system. In the beginning we had some problems trying to figure out the optimal hyperparameters for the model, in the end, after multiple trial and errors we arrived at the configuration described in section 2. The performance of the system in the training phase can be seen in Table 3 where we submitted 2 runs and got two very different results, the first run consisted of the model trained for 10 epochs and the 3rd run was trained for only 5, it can clearly be seen that the jump in performance is quite significant; the third run was ultimately the algorithm that was used for the official results.

Run	Accuracy	Epochs
1	54.0821%	10
3	92.2316%	5

Table 3: Results in training phase

In the main competition the model ranked 17th out of 45 submission with an accuracy of 89.1% (Table 4). This result is in line with what we were able to predict from tests and what we expecting to score (with a minor difference, of course).

No.	Team	Accuracy(Place)	Submissions
19	UAICS	89.1% (17)	3

Table 4: Official results.

We consider that the results are quite good and that the system is on the right path, work on besting our results has already begun, we plan on training on multiple GPUs in order to reduce train time(as this was a big problem for us).

3.1 Error analysis

Although the accuracy is encouraging after careful analysis it has been concluded that the model alone cannot reach a higher accuracy. We now believe that the model needs external information in order to better infer sentence correctness.

Our future plan is to include a module from which the model can access world knowledge in different ontologies. By doing so we will solve the cases where prior information is needed, such as in table 5 where it can clearly be seen that it is almost impossible for the system to have such knowledge about the things in question.

[12] https://colab.research.google.com/

id.	Sentence 0	Sentence 0
4389	Mark Twain is a scientist.	Stephen Hawking is a scientist.

Table 5: Complex reasoning example.

4 Conclusion

To conclude, in this paper we proposed a model to solve the Task 4 challenge which deals with evaluating a systems ability to detect sentences that are against commonsense. We used BERT as the main tool to help us obtain good results, we ranked 17th out of 45 submission with an accuracy of 89.1%. As future work, besides the ones already mentioned, we are currently trying to improve the systems accuracy by including an external ontology in order to better infer whether a phrase is against common sense or not.

Acknowledgements

This work was supported by project REVERT (taRgeted thErapy for adVanced colorEctal canceR paTients), Grant Agreement number: 848098, H2020-SC1-BHC-2018-2020/H2020-SC1-2019-Two-Stage-RTD.

References

Lucia Georgiana Coca, Ciprian-Gabriel Cusmuliuc, and Adrian Iftene. 2019. Checkthat! 2019 UAICS. In Linda Cappellato, Nicola Ferro, David E. Losada, and Henning Müller, editors, *Working Notes of CLEF 2019 - Conference and Labs of the Evaluation Forum, Lugano, Switzerland, September 9-12, 2019*, volume 2380 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Jeremy Howard and Sebastian Ruder. 2018. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Vid Kocijan, Ana-Maria Cretu, Oana-Maria Camburu, Yordan Yordanov, and Thomas Lukasiewicz. 2019. A surprisingly robust trick for winograd schema challenge. *CoRR*, abs/1905.06290.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'12, page 552–561. AAAI Press.

Trieu H. Trinh and Quoc V. Le. 2018. A simple method for commonsense reasoning. *CoRR*, abs/1806.02847.

Cunxiang Wang, Shuailong Liang, Yue Zhang, Xiaonan Li, and Tian Gao. 2019. Does it make sense? and why? A pilot study for sense making and explanation. *CoRR*, abs/1906.00363.

Cunxiang Wang, Shuailong Liang, Yili Jin, Yilong Wang, Xiaodan Zhu, and Yue Zhang. 2020. SemEval-2020 task 4: Commonsense validation and explanation. In *Proceedings of The 14th International Workshop on Semantic Evaluation*. Association for Computational Linguistics.