Assignment 2

Antonio Politano, Enrico Pittini, Riccardo Spolaor and **Samuele Bortolato**
Master's Degree in Artificial Intelligence, University of Bologna
{ antonio.politano2, enrico.pittini, riccardo.spolaor, samuele.bortolato }@studio.unibo.it

Abstract

In this work we propose a model for addressing the QaA task which consists in two pre-trained transformer-based modules: the token importances extractor and the encoder-decoder. The training of such model is based on the combination of two different losses, with the usage of a teacher-forcing-like strategy. The results are satisfactory with respect to the average SQuAD F1 score.

1 Introduction

Question Answering (QA) is the task consisting in generating answers for questions referring to the passages containing the needed information. Optionally, also the history of previous question-answer turns can be used for producing the answer.

In our work, we use a model which consists of two modules: the **Tokens Importances Extractor** (TIE) and the **Encoder-Decoder** (ED). The first module computes an importance score in $[0, 1]$ for each passage token, representing the likelihood that the token is in the span of the passage containing the answer. The encoder-decoder will then use these importances as additional information to generate the answer. The reason of following this approach is to help the encoder-decoder in finding the interesting information in the passage, since it can be very long. Both modules are built from a pre-trained transformer-based architecture.

The dataset taken into account is the CoQA dataset, using the given training-test split and further splitting the training into training-validation, with proportions $0.8 - 0.2$. The unanswerable questions are deleted.

Two different pre-trained transformer-based architectures have been used, namely **DistilRoBERTa** and **BERTTiny**. For both models, three different random seeds have been tested. Overall, 12 different experiments have been run, since there is the possibility to consider or not the conversational history.

2 System description

Our model consists of two modules: the **Tokens Importances Extractor** and the **Encoder-Decoder** (Fig 1).

- The TIE is a pre-trained transformer-based encoder, with on top a linear layer and a sigmoid. Basically, the encoder computes the contextual embeddings of the input tokens and then the linear layer computes the scalar importances scores out of them.
- The ED is a pre-trained transformer-based encoder-decoder modified to make it accept the tokens importances as second input of the encoder. The importances are injected inside the model by passing them through a linear layer and adding the result to the input of each encoder block. We have chosen to use a different linear layer for every block of the encoder.

Both modules are built from a pre-trained transformer-based architecture, either `DistilRoBERTa` or `BERTTiny`, taken from Hugging Face (Wolf et al., 2020). For implementing the ED, the Hugging Face encoder-decoder has been modified in order to take in input also the tokens importances and using them in each encoder block. For making this modification, the Hugging Face code has been taken and adapted for our needs, both for `DistilRoBERTa` (HuggingFace, 2020b) and `BERTTiny` (HuggingFace, 2020a).

The used loss function is the sum of two losses.

- The first loss is about the goodness of the TIE. It measures the difference between the true span in the passage and the importances over the passage assigned by the extractor. The loss is a weighted binary crossentropy, where the weights account for the imbalanced classes.

The target label of each token in the passage is 1 if it is in the span, 0 otherwise (Fig 2).

- The second loss is about the goodness of the ED. It measures the difference between the generated and the true answer. This is measured using the same standard loss function of the encoder-decoder: the cross-entropy loss.

It's worth highlighting that the gradient of the second loss also flows through the TIE, potentially teaching it to do more than just finding the span.

In order to optimize the training for the 3 epochs we had available, instead of passing to the ED the predicted importances we passed a linear combination between the prediction and the target (teacher forcing). This allows the ED to learn even when at the beginning the TIE is still not able to identify the span correctly. Training the ED only using perfect importances could not prepare the network adequately for validation and test where it actually has to use the predicted importances, so instead we gradually turned off the supervision following a cosine schedule (Fig 3). Intuitively, we first let the DE understand how it's supposed to use the importances by giving it the reference ones, and then gradually teach it to use those predicted by the TIE by turning off the supervision.

3 Experimental setup and results

Regarding the training procedure, the Adam optimizer has been used, with learning rate $5 \cdot 10^{-5}$, with batch size 8 and 3 epochs. Three different random seeds have been set and also whether to use or not the conversational history has been tested. Overall, 12 different experiments were run and evaluated on the validation and test sets, by using the average SQuAD F1 score (robinjia, 2018) on all the instances. Table 1 sums up the results.

4 Discussion

The best `DistilRoBERTa` configuration according to the validation score is the one with seed 2022 and using the QaA history, while the best `BERTTiny` configuration is the one with the seed 2022 and not using the QaA history.

The model `DistilRoBERTa` is particularly good even in the worst cases. The less satisfactory results refer to the source *McTest*, while the ones of the other sources are less serious. Some errors are just an interpretation which is correct, although different from the expected one. The majority of the errors are either a wrong choice between multiple possibilities or incorrect answers that are still valid out of context for the given questions.

The model `BERTTiny` is particularly bad with respect to `DistilRoBERTa`. This is expected, since the former model is extremely lightweight and less complex than the latter. The TIE seems to be pretty inconsistent and unreliable for what concerns the worst errors. Most of the given answers are completely unrelated to the questions even out of context and the model often wrongly predicts "yes" as an answer. It is interesting to notice that for the given cases "yes" is the only present choice between "yes/no" answers.

During training, both the models showed a slight increase of the training loss during the third epoch (Fig 4) when the teacher forcing supervision is being completely removed, suggesting that finding the relevant information in the passage is the hardest task and the limiting factor for the performance.

For both models, most of the errors are obtained from questions that are pondered in an already established conversation.

5 Conclusion

Overall the `DistilRoBERTa` model obtains quite good results according to the average SQuAD F1 scores. Even considering the "worst" errors, the model produces logically coherent answers. On the other hand the `BERTTiny` model, as expected due to its much smaller dimension, has worse results and some of the generated answers are totally off the context of the questions.

In order to obtain better results increasing the training epochs could be a determining factor.

	`DistilRoBERTa`				`BERTTiny`			
	Validation SQuAD F1		Test SQuAD F1		Validation SQuAD F1		Test SQuAD F1	
	No History	History	No History	History	No History	History	No History	History
seed 42	0.48132	0.49752	0.49185	0.51392	0.23645	0.22185	0.23854	0.23488
seed 2022	0.47511	0.50281	0.49350	0.52342	0.24117	0.23626	0.24610	0.24361
seed 1337	0.44120	0.48739	0.44890	0.51066	0.24062	0.23203	0.25225	0.23502

Table 1: Average SQuAD F1 score evaluation for different seeds with and without QA History

6 Links to external resources

Link to GitHub repository.
Link to model weights folder.

References

HuggingFace. 2020a. transformers/models/bert/modeling_bert.py.

HuggingFace. 2020b. transformers/models/roberta/-modeling_roberta.py.

robinjia. 2018. Official evaluation script for squad version 2.0.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Appendix

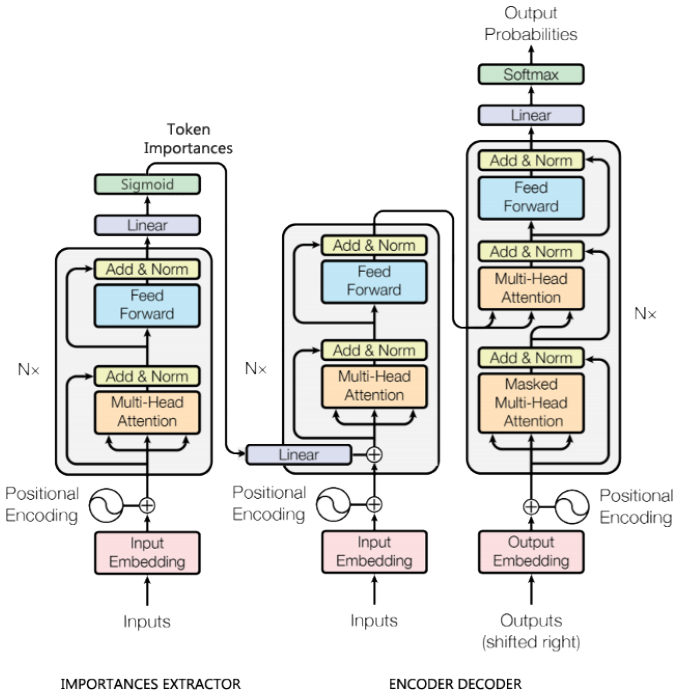

Figure 1: Model architecture

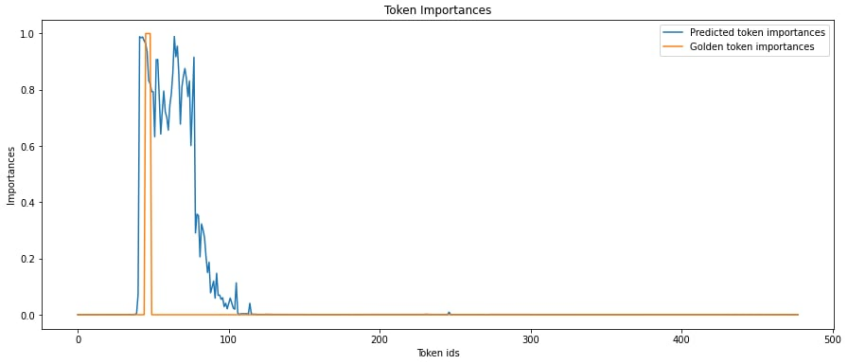

Figure 2: The token importances produced by the extractor for a certain sample versus the true passage span.

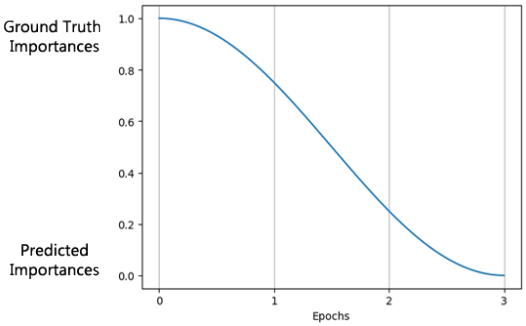

Figure 3: Teacher forcing schedule

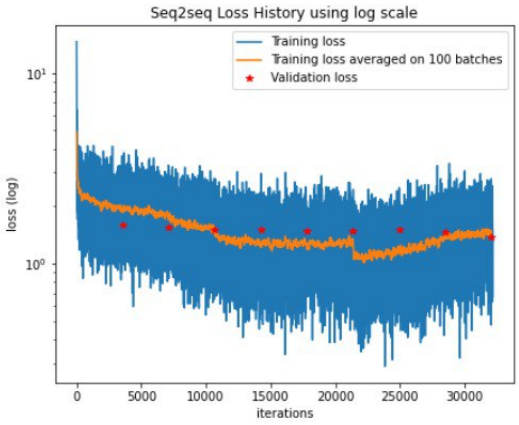

Figure 4: Train and validation losses (log scale)