Machine Comprehension

Anastasiia Shalygina
nshal3879@gmail.com

June 7, 2019

Abstract

Machine comprehension or, in other words, answering a query on the basis of the given context paragraph have gained popularity over the natural language processing community within the last few years. This paper describes experiments with AllenNLP[1] machine comprehension model, a reimplementation of the Bi-Directional Attention Flow model (BiDAF), described in [1] by Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi and Hananneh Hajishirzi, which is widely used as a baseline that achieved state-of-art results on the SQuAD dataset[2] in 2017.

1 Introduction

1.1 Model description

Bi-Directional Attention Flow is a hierarchical multi-stage architecture for modelling the representations of the context paragraph at the different levels of granularity [1]. Machine comprehension model described in [1] consists of 6 layers: Character Embedding Layer, Word Embedding Layer, Contextual Embedding Layer, Attention Flow Layer, Modeling Layer and Output Layer. All the layers can be seen in the Figure 1.

First three layers of the BiDAF model applied to both the context and the query. Character Embedding Layer maps each word to a high dimensional vector-space using character-level CNNs. Word Embedding Layer maps each word to a high-dimensional vector space using pretrained word embeddings (in the original paper [1] GloVe pretrained word embeddings used for this purpose). Then, in the Contextual Embedding Layer LSTM network used on the top of the embeddings produced by the previous two layers to model temporal interactions between words. Thus, the first three layers of the model are extracting features from the query and the context.

The next layer is the Attention Flow Layer. It couples the query and context vectors and produces a set of query-aware feature vectors for each word in the context. Attentions

[1]https://allennlp.org/models
[2]https://rajpurkar.github.io/SQuAD-explorer/

are computed in two directions: from the context to query and from the query to the context. Both of these attentions are derived from a shared similarity matrix, S, between the contextual embeddings of the context and the query. S then used to obtain attentions and attention vectors in both directions. Attention vectors and contextual embeddings combined together and passed to the Modeling Layer.

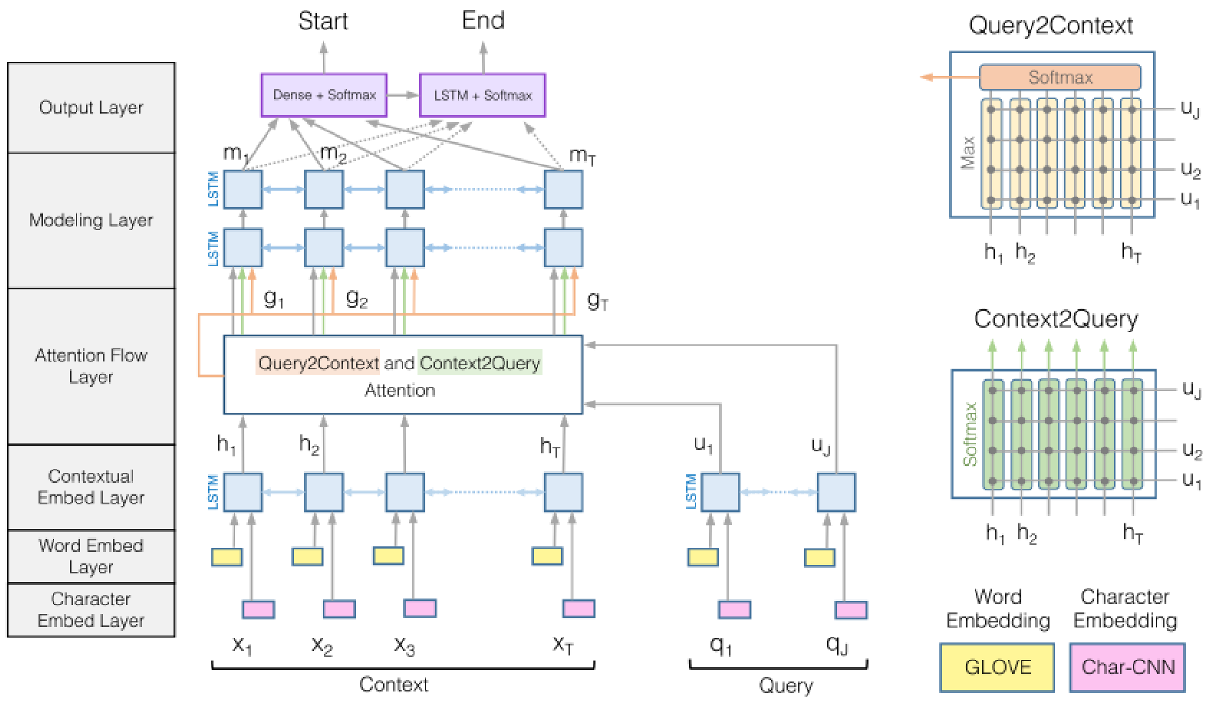

Figure 1: BiDAF Model [1]

Modeling Layer is two layers of bi-directional LSTM. The output of the Modeling Layer captures the interaction among the context words conditioned on the query. This is different from the contextual embedding layer, which captures the interaction among context words independent of the query [1]. Finally, the outputs of the Modelling Layers goes to the Output Layer which produces an answer to the query.

1.2 Baseline model performance

As a first part of the project, I retrained the BiDAF baseline model. I trained it for 20 epochs with patience 10 as in the original configuration file. Training and validation F1-scores and losses have shown on the Figures 2 and 3 respectively. It can be seen from the plots that the best epoch was 5th epoch and then model started to overfit. Since the patience parameter was 10, training stopped after the 15th epoch. After the training, I ran an evaluation and received F1-score of 0.768 on the evaluation set.

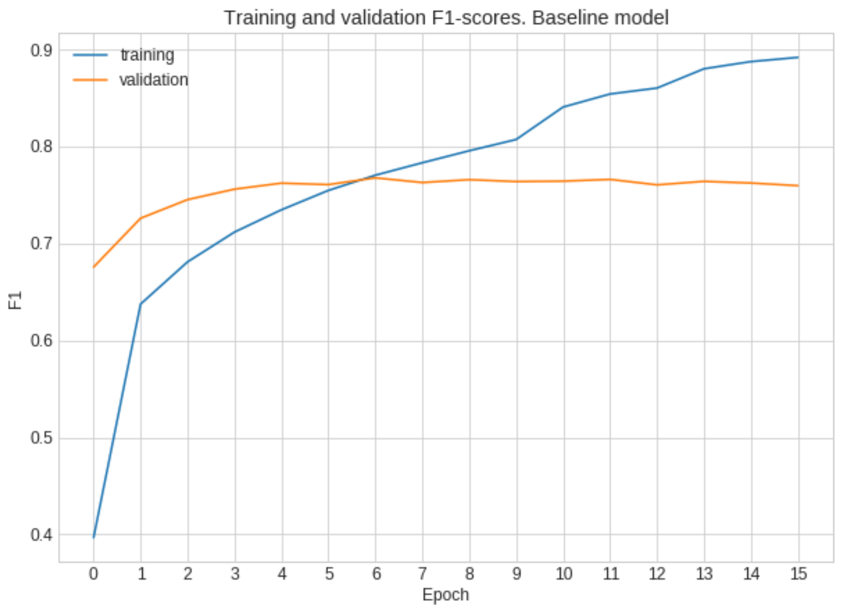

Figure 2: Training vs validation F1-score

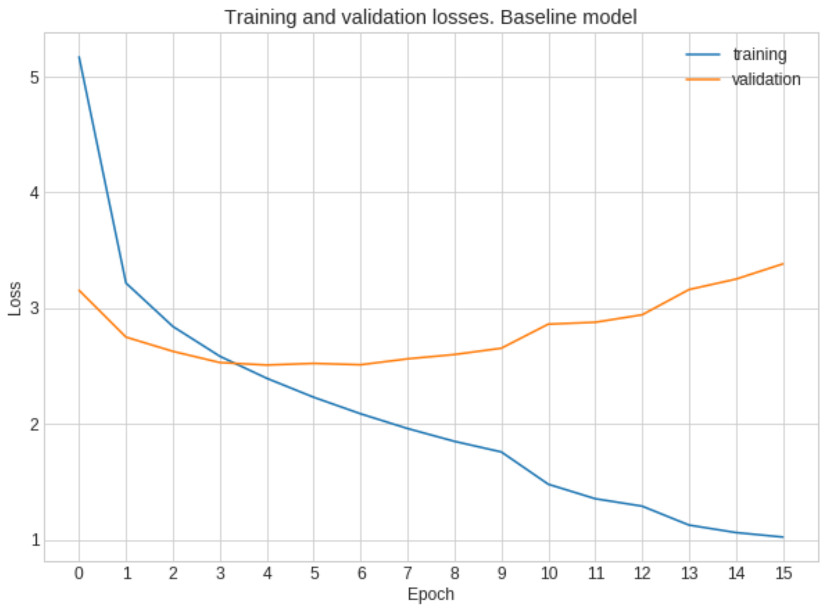

Figure 3: Training vs validation loss

2 Experiments

In this project, I tried to do some experiments and modifications of the original BiDAF model in order to improve its performance. Code, configuration files and logs can be found in the project repository[3].

2.1 FastText embeddings

As for the first experiment I replaced GloVe pretrained embeddings which are used in the original implementation with FastText pretrained embeddings. FastText is a word embedding method that is an extension of the word2vec model. Instead of learning vectors for words directly, FastText represents each word as an n-gram of characters. This helps capture the meaning of shorter words and allows the embeddings to understand suffixes and prefixes. Also, FastText embeddings have 300 dimensions, whereas GloVe embeddings used in the baseline model are 100-dimensional. I assumed that the differently constructed embeddings with a higher dimensionality could give an improvement in the results.

I have trained the BiDAF model with FastText embeddings also with a limit of 20 epochs and patience 10. After the evaluation, this model gave F1-score of 0.772 on the evaluation set. Plots of the training and validation F1-scores and losses can be seen below on the Figures 5–8.

2.2 ELMo embeddings

For the second experiment, I decided to include ELMo embeddings to the model. In the paper "Deep contextualized word representations" [2] authors establish that the addition of ELMo embeddings to the BiDAF model can give an improvement of F1-score by about 5%.

It is concluded in [2] that in order to add ELMo to the supervised model, we first need to freeze the weights of the biLM and then concatenate the ELMo vector with a context-independent token representation and then pass the enhanced representation into the task RNN. In the case of BiDAF model, this means that the ELMo representation is concatenated with inputs at the Character Embedding Layer and the outputs of the Contextual Embedding Layer and it can be interpreted as it is shown on the Figure 4.

For my experiment, I only managed to apply ELMo in the Word Embedding Layer, after the GLoVE embeddings. ELMo makes training much longer, so I decided to use this modification and compare the result with the results from other experiments. Modified configuration file with ELMo embeddings can be found in the project repository.

[3]https://github.com/ShalyginaA/allennlp-machine-comprehension

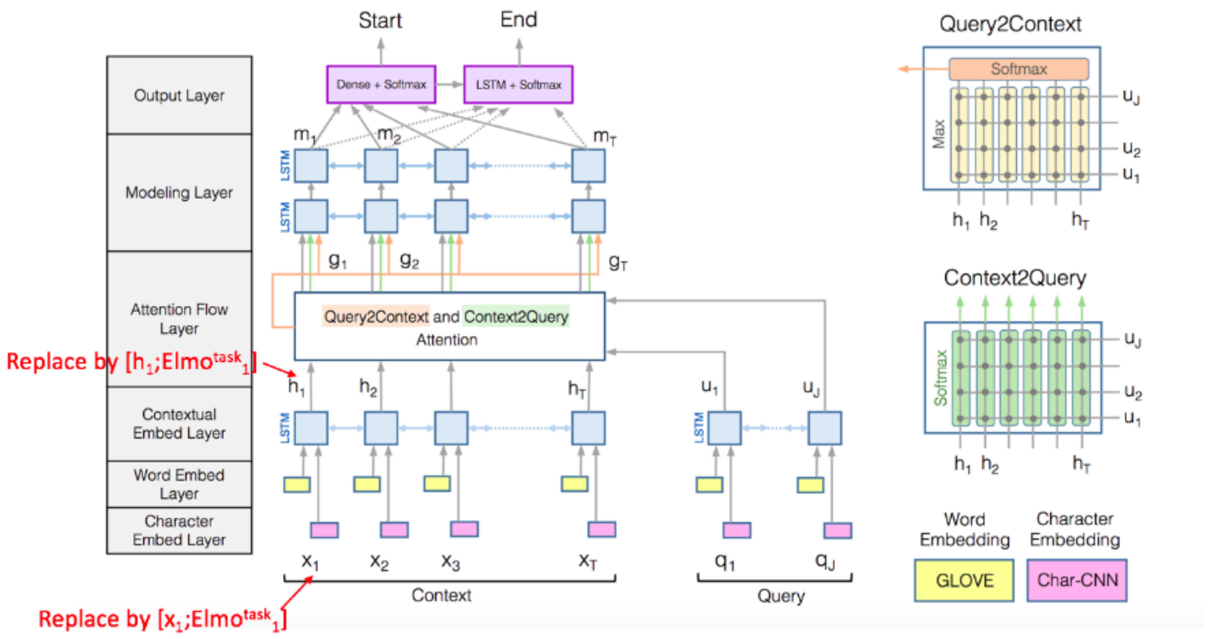

Figure 4: BiDAF Model with ELMo embeddings

I trained and evaluate the described model and observed a significant improvement of F1-score.

2.3 Regularization and Cosine Matrix Attention

In the last experiment, I decided to pay attention not only for configuration files but for the code and discovered that for the calculation of similarity matrix S mentioned in the Introduction LegacyMatrixAttention[4] with a linear similarity function used.

As it mentioned in the AllenNLP documentation, LegacyMatrixAttention module should be considered deprecated as it uses much more memory than the newer specialized MatrixAttention module. Also, it is a few other modules can be used to calculate a similarity matrix. Thus, I decided to replace LegacyMatrixAttention with a CosineMatrixAttention which computes attention between two matrices using cosine similarity.

Also, I added a regularisation to the model using RegularizerApplicator[5] module in order to calculate the regularization penalty during training which could potentially improve the results.

[4]https://allenai.github.io/allennlp-docs/api/allennlp.modules.matrix_attention.html
[5]https://allenai.github.io/allennlp-docs/api/allennlp.nn.regularizers.html

I trained the modified model for the same number of epochs and with the same patience parameter as in the previous experiments and received and F1-score of 0.727 on the evaluation set.

2.4 Comparison of the models

Plots of training and validation F1-scores and losses for all the described models presented on the Figures 5–8 in order to compare performances of the models. As it can be clearly seen, the addition of ELMo embeddings to the BiDAF model gives a significant improvement in the F1-score on both training and validation sets and also faster decrease of the loss. The worst performing model is the one with the regularization and cosine similarity. Replacement GLoVe with FastText almost not changing the model performance.

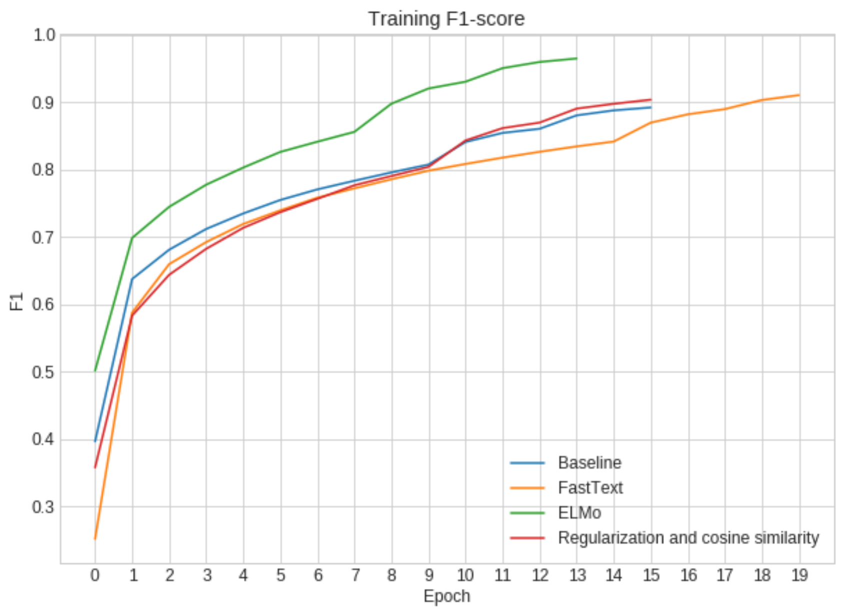

Figure 5: Training F1-scores of different models

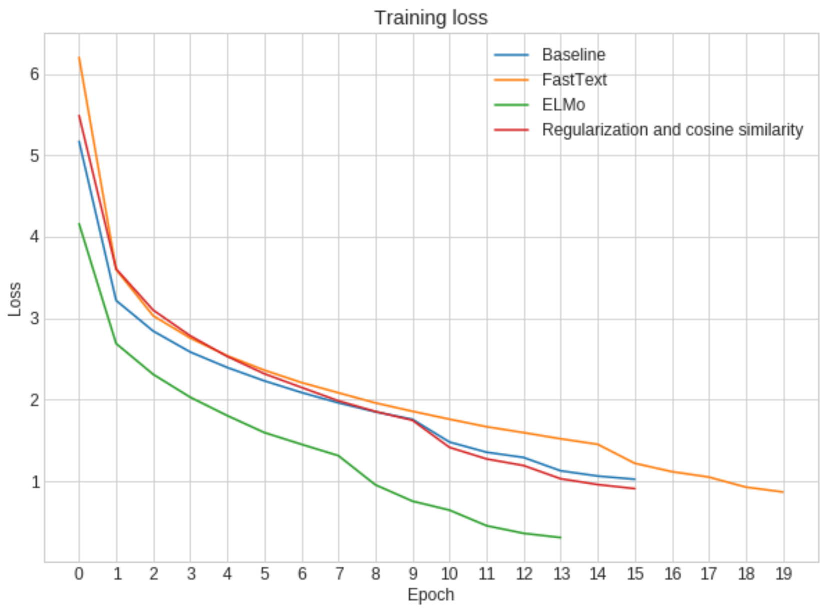

Figure 6: Training losses of different models

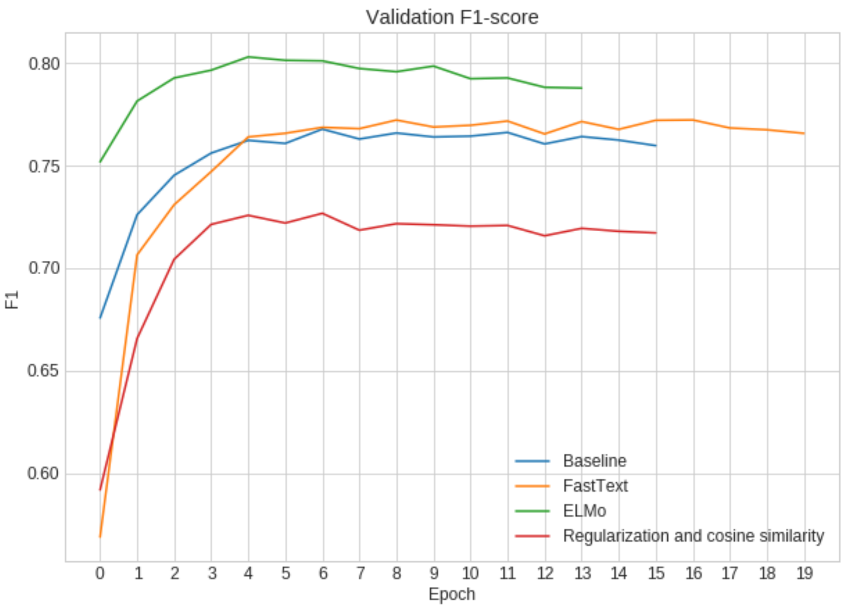

Figure 7: Validation F1-scores of different models

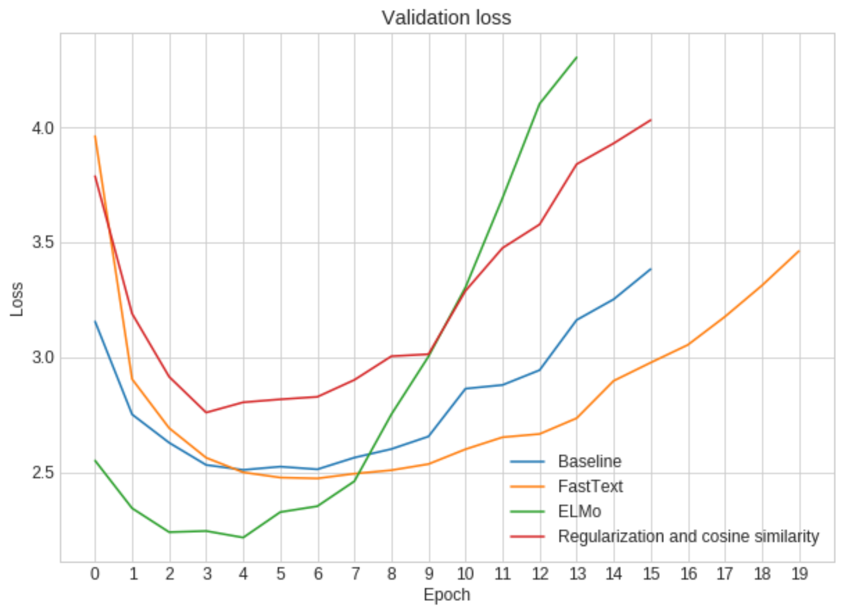

Figure 8: Validation losses of different models

F1-scores on the evaluation dataset for all the described models have shown in Table 1. The best F1-score on the evaluation set gives BiDAF model with the addition of ELMo embeddings. It outperforms the baseline F1-score by $\sim 5\%$.

Model	**F1 score**
Baseline,	0.768
FastText,	0.772
ELMo,	0.803
Reg. + Cosine	0.727

Table 1: F1-scores on the evaluation set

3 Discussion

I analyzed the performance of the models described above. The best modification of the baseline model in order to achieve higher F1-score was an addition of ELMo embeddings after the GLoVe embeddings. However, this modification increases training time significantly. Since the BiDAF model is not very memory-efficient and the model training takes a lot of resources, it can be a huge disadvantage. Thus, I would rather use the baseline model in the cases when the result needs to be obtained faster. But if it is enough of time and computational power, then the modified model could be useful.

4 Conclusion

In this project, I introduced a few experiments with AllenNLP BiDAF model modifications. I trained and evaluated these models on the SQuAD dataset and compared the results using the original model as a baseline. For this purpose, I used visualizations of F1-scores and losses for the training and validation sets as well as evaluation scores. Improvement of the baseline results was reached by the addition of ELMo pretrained embeddings to the original BiDAF model.

References

[1] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi and Hananneh Hajishirzi *Bi-Directional Attention Flow for Machine Comprehension.* arXiv:1611.01603v6, 2018

[2] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee and Luke Zettlemoyer *Deep contextualized word representations.* arXiv:1802.05365v2, 2018