Experiments in Embeddings with BiDAF

Stanford CS224N Default Project

Sterling Alic
Department of Computer Science
Stanford University
salic@stanford.edu

Miles Zoltak
Department of Computer Science
Stanford University
mzoltak@stanford.edu

Abstract

Deep learning has revolutionized the field of natural language processing (NLP) in recent years. One focus within this field is neural question answering systems, which we explore in this paper. Our project extends the baseline BiDAF model to integrate character-level embeddings and fine-tuning embeddings. We then adjust hyper-parameters such as dropout, batch size, and hidden layer size to improve our model performance. We find that character-level embeddings in our model imporove performance over the baseline with EM and F1 scores of 61.032 (+3.277) and 64.070 (+2.842) respectively. We also analyze the strengths of our extensions to the BiDAF model by question-word and outline directions for future work with embeddings and additional input features.

1 Key Information to include

- Mentor: Benjamin Newman
- External Collaborators (if you have any): N/A
- Sharing project: Yes

2 Introduction

Question answering (QA) has been called the strongest possible demonstration of natural language understanding. As such, it has been a task within Natural Language Processing that has seen a great deal of research and advancement. The goal of a (closed-domain) QA system is to process some query and extract a passage from a relevant context paragraph that contains the answer to the query.

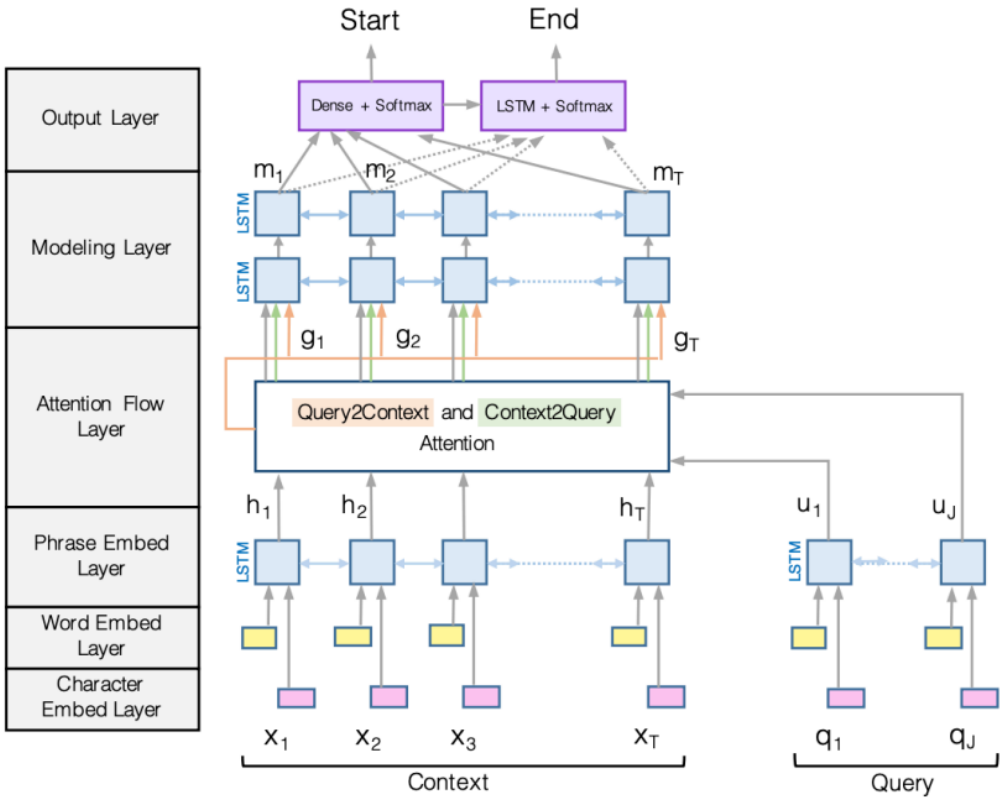
One of the most famous datasets in QA is the Stanford Quesation Answering Dataset (SQuAD). SQuAD consists of 100k annotated triples crowdsourced from human workers. It has long been the standard by which QA systems are developed, trained, tested, and evaluated. Among several high-performing systems, one of the most famous is the Bi-Directional Attention Flow model, or BiDAF, which uses attention to reduce information loss when the context paragraph is summarized [1]. It is bi-directional because attention is computed from context to query and back from query to context. Without any extension, this model achieved a score of 77.3 on SQuAD.

In this project, we attemp to explore variations on the BiDAF model for QA. We implement some well-defined extensions of BiDAF such as character-level embeddings as well as substituting LSTMs for GRUs. We explore other tweaks to the system that are not as clear-cut via hyper-parameter search to find alternative values for parameters such as dropout rate, batch size, and size of the hidden layer.

3 Related Work

The baseline model we extend in this project is based on the BiDAF model from Seo et al., 2017. The architecture for this model can be seen below in Figure 1. Note that the baseline model does

not implement character-level embeddings, so the word embeddings are passed directly to the bi-directional attention layer.



(Seo et al., 2017): Bidirectional Attention Flow for Machine Comprehension

Figure 1: Model architecture of the Bidirectional Attention Flow (BiDAF) model, the architecture for the baseline that we built upon in our original system.

Other systems that have submitted to the SQuAD leaderboard have also fine-tuned the initial embeddings in their approach for task-specific data, such as movie data and medical data on question-answering [2]. This system also used Part-of-Speech (POS) and Named-Entity Recognition tagging as additional features, an approach that we didn't have time to implement, but outline as a next step for us in our future work section.

4 Approach

4.1 BiDAF Model

Our BiDAF model differs from the baseline model in that we added character-level embeddings and we trained using various different hyper-parameters. More specifically, we begin by passing the character-level embeddings through a concurrent neural network with max-pooling, and then concatenating the output to the word-embeddings before training.

This new "phrase-embedding" is then passed through a highway network before using an LSTM to encode these phrase embeddings. After being encoded, we use bi-directional attention to allow the attention at each timestep to flow through the attention layer and then into the modelling layer, which is yet another LSTM. Finally, we perform a linear transformation on the output of the modelling layer and use a softmax to determine the position of the predicted answer in the context paragraph.

We initially experimented with using Gated Recurrent Units (GRUs) instead of LSTMs as a different kind of RNN, but in the end we moved forward with the LSTM. We cover the reasoning behind these changes in section 5.3 Experimental Details.

Our final model architecture can be seen in Figure 2.

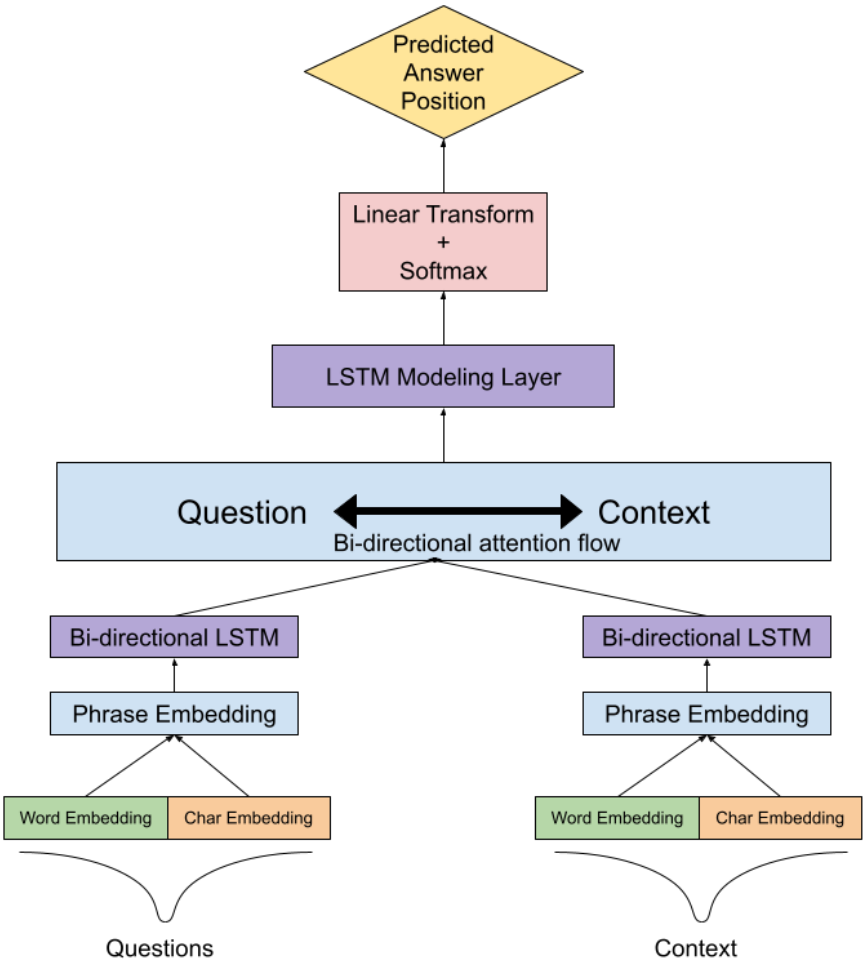


Figure 2: Our system, which consists of the BiDAF model architecture combined with character-level embeddings.

4.2 Character-Level Embeddings

Our main approach was to add character-level embeddings to the BIDAF model in hopes of improving upon the baseline performance [1]. The purpose of using character-level embeddings is to allow the model to learn from morphological patterns as well as handling out of vocabulary (OOV) words. In other words, character-level embeddings help us catch and predict based on words that we may not have seen during training but will still help us answer questions.

More specifically, we concatenate these embeddings with the word level representations and feed it into the highway network. The purpose of this is to allow the highway network to decide which part of the concatenated embedding is more useful for understanding the text. For example if the model came across the word "reimploded" the GloVe embedding for the word itself would be unhelpful because "reimploded" is not a real word. But it *is* made up of meaningful morphemes and roots

like "re-", "-ed", and "implode". The character-level embeddings would still communicate some amount of meaning in the word, and the highway network tells the model to pay more attention to the character-level information than that of the word-level.

4.3 Fine-tuned GloVe embeddings

One approach we took to try to improve the performance of our model was to use the Mittens python module to finetune our GloVe embeddings [3]. To do this, we took a fraction of the train dataset and created a set of all the words that appeared in context paragraphs. We used this set to determine words that were found in context paragraphs that were missing from GloVe, and then fitting a new adjusted GloVe model to those out of vocabulary (OOV) words. The purpose of this, as stated in [3]the original paper, is to achieve "faster learning and better results" from our model.

5 Experiments

5.1 Data

Our dataset used was the SQuADRUn [4] dataset, also known as SQuAD 2.0. It is created from the original SQuAD dataset with the addition of 50,775 unanswerable questions about the same paragraphs from the SQuAD dataset. SQuAD itself has 100,000 question-answer pairs over500 articles. The total size of SQuADRUn then is roughly 151,000 questions (2:1 answerable to unanswerable), with over 500 articles. More specifically, the dataset contains triples of (question, context, answer). We input a question and a paragraph context pair to our model. Then the model predicts an answer as output (or an empty string to signify that the question isn't answerable).

Question: Why was Tesla returned to Gospic?
Context paragraph: On 24 March 1879, Tesla was returned to Gospic under police guard for not having a residence permit. On 17 April 1879, Milutin Tesla died at the age of 60 after contracting an unspecified illness (although some sources say that he died of a stroke). During that year, Tesla taught a large class of students in his old school, Higher Real Gymnasium, in Gospic.
Answer: not having a residence permit

Figure 3: An example entry from the SQuAD dataset, including a question, context paragraph from Wikipedia, and ground truth answer.

5.2 Evaluation method

We evaluate our model performance using the F1 score and Exact Matching (EM) metrics. Exact Matching is a binary measure (0 or 1) of if the model prediction is an exact match of the ground truth answer. For example, if the ground truth answer *Paris France* and the model predicts *France*, then the EM measure would be 0. The F1 score, on the other hand, is the harmonic mean of precision and recall, and it is a less strict metric. It scores the previous example model output of *France* as having 100% precision and 50% recall. The precision is 100% because the answer is a subset of the ground truth answer. The recall is 50% because only one of the two words in the ground truth answer is in the model's predicted answer. Therefore, the F1 score is approximately 67%.

5.3 Experimental details

We built off an implementation of BiDAF from the Default Project starter code. We first experimented with replacing the model's default LSTM with a Gated Recurrent Unit (GRU), which is reported to be optimized for efficiency. We confirmed this in our experiments, as the baseline with the GRU model trained faster than the LSTM baseline model [INSERT HOW MUCH FASTER]. However, as we received worse performance with the baseline, we proceeded with using the LSTM, we proceeded with focusing on tuning other hyperparameters.

5.4 Hyperparameter Tuning

We tuned the following hyperparameters: dropout probability, hidden size, and batch size. We performed a random search over a range of hyperparameters, and our results of these experiments are shown in Figure 3. We chose hidden size and batch size to increase the model's capacity and able to pick up trends in the dataset, while we chose to tune the dropout probability to regularize against the model overfitting to the data.

Moreover, since the ADADELTA optimizer has a dynamic learning rate that has been shown to be resilient to in a variety of situations, we chose to refrain from tuning the learning rate [5]. We also mostly refrained from tuning epochs, except in one case of early stopping when the F1 score showed little to no improvement on the evaluation set during training and due to limited time.

5.5 Results

We report the following results on the leaderboard, as shown in Figure 4, using the pre-contextualized embeddings (PCE track). We saw that the character embeddings improved upon the baseline performance, achieving an EM of 61.03 and an F1 score of 64.07 on the dev set. However, our hyperparameter search did not yield any additional gains in performance. Thus, we see the promise in using pretrained embeddings in a system designed to perform well on SQuAD. One limitation of our approach is that we could have done a more thorough hyperparameter search given more time/resources. Therefore, future researchers can build upon our work by performing a larger search in hopes of boosting model performance. Our final submission to the test set leaderboard yielded an EM score of 60.524 and an F1 score of 64.296.

Model	Dropout	Hidden Size	Batch Size	Epochs	EM	F1
Baseline GRU	0.2	100	64	30	59.872	63.168
Baseline LSTM	0.2	100	64	30	57.755	61.228
LSTM + Char Embed	0.2	100	96	30	59.30	62.81
LSTM + Char Embed	0.25	100	128	30	57.22	60.74
LSTM + Char Embed	0.15	150	64	20	60.11	63.35
LSTM + Char Embed	0.15	100	64	30	59.44	63.04
LSTM + Char Embed	0.2	100	64	30	**61.032**	**64.070**

Figure 4: The results of our hyperparameter search. We performed a random hyperparameter search over five hyperparameter combinations using the weights and biases (WB) library.

5.6 Analysis

We performed further analysis of our best model, breaking down the performance of the LSTM model with character embeddings and default parameters on question words.

We hypothesized that this could be due to two factors: (1) the frequency of question words in the training data and (2) the complexity of questions containing certain question words. We first thought intuitively that the more frequent question words in the training data would have higher scores, which we did see for question words like "who" and "what," the top two most frequent question words in the training dataset. We also saw. However, we still saw high performance for moderately frequent question words like "which." This led us to intuit that one of the reasons for the difference in performance across question words was because of the complexity of the question, where the model performed worse on more complex, while performing better on less complex questions. We saw this pattern confirmed in the data, as the model performed significantly worse on open-ended question words like "how" and "why." Meanwhile, it performed the best on question words that are associated with closed-ended questions like "which" and "when."

6 Conclusion

Our main findings were that many of the alterations we made to the baseline model did not help our performance. While gated recurrent units (GRUs) would have ideally sped up our training, they did so at the cost of accuracy and we were not able to move forward with them. Additionally, our adjusted hyperparameters did not give us any performance gains either.

Question Word	EM	F1
Who	60.428	62.261
What	60.783	63.944
Where	61.029	64.477
Which	66.667	69.957
When	68.592	69.780
Why	49.425	54.749
How	56.616	61.056
Other	38.461	42.749

Figure 5: Performance of our best model (LSTM + character embeddings) broken down by question word.

The primary limitation for us was time. We were not able to implement the finetuning with mittens in time to actually test their performance, this was an adjustment we were particularly eager to test. Additionally, the number of dimensions in the GloVe embeddings may have also contributed to the long training time along with the slowness of the finetuned mittens embeddings. Exploring GloVe embeddings with fewer dimensions may have alleviated this issue. Ironically, we did not have enough time to explore this possibility either.

The character-level embeddings remain the highlight of our model extension. This was expected as it is a well-documented and effective method of extending the BiDAF model.

7 Future Work

While we did finteune the dataset using Mittens, we did not get the chance to test with these new embeddings and see if there were any performance gains. In future explorations we would like to incorporate these features along with part of speech (POS) tags to help the model make better sense of given information to improve performance. Another improvement we would like to do is implement early stopping, such that runs in hyperparameter tuning that don't improve our evaluation metrics for a certain number of epochs terminate early.

References

[1] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[2] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[3] Nicholas Dingwall and Christopher Potts. Mittens: an extension of GloVe for learning domain-specialized representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 212–217, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[4] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[5] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.