# Experimenting with BiDAF Embeddings and Coattention

Stanford CS224N Default SQuAD Project

**Chris Tan**
Department of Computer Science
Stanford University
ctan17@stanford.edu

**Makena Low**
Department of Computer Science
Stanford University
makenal@stanford.edu

## Abstract

Our project is motivated by the task of question answering, which is a natural application of language models that is also helpful in evaluating how well systems understand the meaning within text. Our primary goal for the project is to improve upon the baseline BiDAF model provided to us on the SQuAD 2.0 dataset, namely by experimenting with character-level embeddings [1], coattention [2], and conditional end pointer predictions [3]. While we weren't familiar with these mechanisms prior to this project, we also think that each of them leads in some way to an intuitive representation of language, linking it to larger aims within the field. We found that character-level embeddings and an Answer-Pointer network greatly elevated the performance of the baseline model, with performance of EM: 58.715 (+58.715) and F1: 62.283 (+62.283) on the test set.

## 1 Key Information to include

- Mentor: Zihan
- External Collaborators (if you have any): none
- Sharing project: none

## 2 Introduction

From search engines and voice assistants to our conversations with one another, the problem of question answering is a natural language task that most of us are already familiar with. Not only does this give us intuition about how to model the task and approach us, but question answering is also a useful task for assessing how well a machine actually understands human language. In order to be able to answer questions accurately, the model must learn some of the meanings and relationships of the words. This makes question answering an pertinent problem to address within the field of natural language processing.

In the past, this problem was limited by the datasets available, namely small human-annotated datasets or larger datasets that that required only limited depths of reading comprehension.[4] With the introduction of the Stanford Question Answering Dataset (SQuAD) in 2016, we can now be more confident that our models are actually attaining a reasonable degree of reading comprehension. Currently, attention networks and transformer models have comprised many of the top performers on NLP tasks, and this has held true for the task of question answering as well.

While we do not propose any novel architectures, our project uses various experiments to better understand the importance and impacts of the layers within an attention network by substituting in other high-performing models and doing a comparative analysis among them. We also tried to be thoughtful in our process of improving upon the baseline and to approach it in a structured manner. Motivating our work was error analysis, understanding the underlying data itself, and contextualizing

the results with respect to the data and the limitations of trying to model something as complex as human language.

## 3 Related Work

Our work relied in large part on existing literatures, given that we compared these various approaches. We looked at various models, and used or considered key elements of many of them. Many of the highest performing approaches utilized attention networks or transformer models. Given the limited timeframe that we had, and being provided the Bidirectional Attention Flow (BiDAF) baseline, we decided to focus primarily on attention networks.

In addition to reviewing the original BiDAF paper by Seo et al.[5], we also reviewed literature around character embeddings, coattention, and a conditional end pointer, as specified in our original plan. Specifically, we looked at Yoon Kim's [1] work, which used a single convolutional layer and max-pooling which performed well on character representations, the work of Xiong et al. [2] on Dynamic Coattention Networks, which outlined the coattention mechanism we implemented as well as a beam search approach to decoding, and Wang and Jiang's [3] work on Match-LSTM and Answer Pointer, from which we derived the answer pointer network.

These works all led to high-performing models on the task of question answering, so our work is an attempt to understand why, and to understand the contributions of each in developing good natural language models. Our approach also focuses on the process of iteratively improving a model in understanding implementing this project as a pedagogical tool.

## 4 Approach

Our approach to the problem consisted of multiple parts, at various parts within the process. We will start by describing the baseline, which we modified as the basis of our experimentation. We were provided the starter code for a Bidirectional Attention Flow (BiDAF) model as outlined in [5], but with only word embeddings implemented. It consists of 5 main layers: an embedding layer, an encoding layer, and attention layer, a modeling layer, and the output (decoding) layer. To give ourselves a place to start from, we began by training the baseline and conducting some initial error analysis to become more familiar with the underlying data and motivate our direction moving forward. From this baseline, we then proceeded to modify it at several points.

At the embedding layer, we tried combining the word embedding with a character embedding. Specifically, we implemented a simple CNN with one layer of 1D convolution followed by a max-pooling layer for sequence lengths of 3, 4, and 5, as outlined by Kim [1], and made the number of feature maps per sequence length another hyperparameter. The outputs of the CNN were concatenated with the word embedding, which served as the input for the two-layer fully connected highway network [6]. In order to keep the input shape to the highway network the same so as to keep it comparable to the baseline, we reduced the dimensions of the word embedding output accordingly. Our reasoning for implementing the character embedding was that the character-level granularity would help capture meanings at a subword level, which could help with generality as well as out-of-vocabulary and unseen words. Furthermore, the starter code made this a natural beginning point.

We also substituted in two alternatives for the BiDAF attention layer, namely a coattention and self-attention. The coattention layer, as suggested by Xiong et al. [2] was particularly interesting to us, as it performed a second-level attention computation in order to fuse the context and query. However, it ended up underperforming relative to the baseline (see Results for more detail). Because of this, we made adjustments to our initial strategy to try to implement self-attention as well, based upon the self-matching mechanism described in R-NET [7]. We thought this approach might also be conducive to improving the baseline, given that self-attention is also the basis for transformer models, which are among the highest performing architectures. Note that we do not use the original form of self-attention as cited in the paper, but rather a vectorized form. For reference, the original paper implements the math in Fig.1.

Unfortunately, using such a model greatly increased our training time which was not feasible within the span of the class. To facilitate training, we instead used the following in Fig.2.

$$s_j^t = \mathrm{v}^{\mathrm{T}} \tanh(W_v^P v_j^P + W_v^{\tilde{P}} v_t^P)$$
$$a_i^t = \exp(s_i^t) / \Sigma_{j=1}^n \exp(s_j^t)$$
$$c_t = \Sigma_{i=1}^n a_i^t v_i^P$$

Figure 1: Original Self-Attention

$$s = w \tanh(W_1 v_{passage} + W_2 v_{passage})$$

Figure 2: Vectorized Self-Attention

The equations were taken from the EdStem post linked here `https://edstem.org/us/courses/3098/discussion/312195`. Note that the implementation of the vectorized form was done on our own, and so is a unique contribution. Implementation for this model was also inspired by this GitHub: `https://github.com/heliumsea/RNet-pytorch/blob/master/models.py`.

Finally, the last substitution layer we implemented was the answer pointer at the output layer, based on the work of Wang and Jiang [3]. Given that the SQuAD task is to predict the start and end pointers for the span in a context paragraph, we based our implementation on their description of the boundary model. The motivation behind this implementing the answer pointer was that this would also provide an improvement by making use of the conditional relationships implied by the span, and thus serve more effectively predict the answer. Implementation for this network was inspired by this GitHub: `https://github.com/laddie132/Match-LSTM/tree/master/models`.

A critical element of our project was our process, or how we tried to take a structured approach to improving the model. While much of the literature we reviewed contained the technical details of implementing novel architectures, we thought we might be able to contribute more to the procedural aspect of building a model. Thus, throughout the project we were intentional about structuring our project around iterative cycles for improving the model and building our own understandings of both the model and the data. Furthermore, we approached our work within context in trying to achieve more accurate representations of language by better modeling the relationships between words. At each stage of the cycle, we would perform error analysis to better understand what the model does well and where it goes wrong, and also allow ourselves to build intuitions about the underlying data being used to train our model. We also adapated our strategy as appropriate as we received feedback in the form of results.

This was particularly salient as we began to approach the final stages of our project. Once we concluded our original experiments with the architecture, including an ablation study, we moved onto finetuning, to see the impacts of adjusting architectures and hyperparameters on performance. While we had initially written a script to conduct our finetuning experiments, we ended up taking a more hands-on approach (also due to time contraints), where we evaluated the change in performance from each adjustment. Our finetuning included training with different numbers of feature maps, epochs, as well as different combinations of architectures. Finally, we used what appeared to be the biggest improvements from each part to construct our final model, and increased the size of the model along with its training time under the assumption that these could only improve performance.

| | Hidden Size | Feature Maps | Epochs |
|---|---|---|---|
| Baseline | 100 | N/A | 15 |
| Character Embeddings | 100 | 10 | 15 |
| Coattention | 100 | N/A | 15 |
| Self-Attention | 100 | N/A | 15 |
| Answer-Pointer | 100 | N/A | 15 |
| Char-Emb + AnsPtr | 100 | 30 | 15 |
| CharEmb V2 | 200 | 30 | 20 |

Table 1: Model configuration details

# 5 Experiments

## 5.1 Data

The data we are using is from the SQuAD 2.0 dataset for the default SQuAD project, split into approximately 90/5/5 train/dev/test splits. Each example has an associated question, context, and answer, where the answer is a span from the context (if the question is answerable).

## 5.2 Evaluation method

The evaluation metrics used are exact match (EM) and F1 score, as stated in the specification document. EM is a measure of whether the answer exactly matches the ground truth label, while F1 is the harmonic mean of precision and recall, using the maximum score evaluated against three human labels.

## 5.3 Experimental details

For every model, we trained on the SQuAD training set provided with a learning rate of 0.5, an ema decay of 0.999, a batch size of 64, a dropout probability of 0.2, a maximum answer length of 15, and an Adadelta optimizer. The models were trained on NV6 Azure VM. See Table 1 for more details.

## 5.4 Results

- Test scores: EM: 58.715 (+58.715), F1: 62.283 (+62.283)

| **Model** | F1 | EM | AvNA |
|---|---|---|---|
| Baseline | 60.24 | 56.61 | 67.65 |
| Character Embeddings | 63.42 | 60.09 | 69.57 |
| Coattention | 57.75 | 54.33 | 64.91 |
| Self-Attention | 52.59 | 49.91 | 60.98 |
| Answer-Pointer | 60.14 | 57.1 | 66.85 |
| Char-Emb + AnsPtr | **63.56** | **60.23** | **69.77** |
| CharEmb V2 | 62.19 | 58.93 | 68.71 |

For character embeddings, the results are better than we expected (see Fig. 3 in the Appendix for dev training curves), as we saw a large improvement in performance across all metrics. However, the improved results make sense, as character level embeddings are more effective at capturing unseen or out of vocabulary words, leading to a higher F1 and EM score. Noteably, even at the early stages of training, our character-embedding model outperforms the baseline. This tells us that our character embeddings approach was a solid addition to the BiDAF baseline.

On the other hand, coattention underperformed more than expected. Note that coattention performs even below the baseline in Fig.3. We attribute the shortcomings to either not implementing it correctly or to reducing the amount of information being passed on to future layers, since its output is half the size of that of BiDAF. It's possible we needed to do a bi-LSTM in the attention layer instead of just the modeling layer, and the smaller amount of information being passed forward (vs BiDAF attention which had 8 times the hidden size, we had 4 time the hidden size) might have reduced our ability to learn.

Self-attention also underperformed quite a bit. This is likely due to the limited nature of single self-attention layer. In transformers, there are generally many self-attention layers, which are also supplemented more parameters. By only using a single self-attention layer, we greatly limited the amount that we could learn. Moreover, because we used a vectorized version of self-attention (see Approaches), we did not have the word-to-word interactions that self-attention typically has, thus further limiting performance. This tells us that perhaps the word-to-word interactions are essential to self-attention in improving our model's performance.

The Answer-Pointer network improved performance a bit, though not quite as much as character embeddings. This makes sense though, as the Answer-Pointer network is not as powerful as character embeddings. Character embeddings changed fundamentally how the model understands language (aka at the character-level, with the ability to recognize OOV words). Meanwhile, Answer-Pointer only conditions the start pointer on the end pointer. While conditioning the ending word based on the starting word makes sense, it is not so groundbreaking as character-level understanding of language.

The Char-Emb + AnsPtr model was our best performing model. This perhaps makes sense as it is the combination of the two features that improved upon our baseline. Moreover, we attempted this run with more feature maps in our character emebeddings layer, which seems to help learning.

Lastly, our CharEmb V2 model performed well, until it started overfitting to the training set. We can see this by seeing how the NLL training loss continues to decrease, while the dev loss increases Fig.4. This makes sense though, as we doubled the amount of hidden layers in this run. We did so because we thought our model would be capable of learning more complex language semantics without overfitting, however this was not the case.

# 6 Analysis

## 6.1 Error Analysis on Baseline

**Strengths**
After training the original baseline BiDAF, we noted a few key strengths of the model. For instance, the model seems to understand related entities. When asked "Question: What tribes supported British?" and "Context: When war broke out, the French used their trading connections to recruit fighters from tribes in western portions of the Great Lakes region (an area not directly subject to the conflict between the French and British), including the Huron, Mississauga, Ojibwa, Winnebago, and Potawatomi. The British were supported in the war by the Iroquois Six Nations, and also by the Cherokee – until differences sparked the Anglo-Cherokee War in 1758." The model correctly responds "Prediction: Iroquois Six Nations" (Answer: Iroquois Six Nations, and also by the Cherokee). Noteable though, the model was not able to pick up "and also by the Cherokee" which may have been remedied with a better BEAM search with further exploration.

The baseline also seemed to perform fairly well on questions regarding numbers such as "When did" or "How many" questions.

**Weaknesses**
The baseline model did not perform well on questions that were phrased Questions that are phrased differently from how the information was presented in the context. For instance, given "Question: What can the exhaust steam not fully do when the exhaust event is insufficiently long?" and the "Context: ... However, as one and the same valve usually controls both steam flows, a short cutoff at admission adversely affects the exhaust and compression periods which should ideally always be kept fairly constant; if the exhaust event is too brief, the totality of the exhaust steam cannot evacuate the cylinder, choking it and giving excessive compression", the model predicts "Prediction: N/A". The correct answer, however, was "Answer: evacuate the cylinder". The baseline model was also weak on answering "why" questions, which intuitively makes sense, as these are generally harder to answer in reading comprehension tasks.

## 6.2 Error Analysis on Best Model

To reiterate, our best model was Char-Emb (with 30 feature maps) + AnsPtr.

### 6.3 Strengths

Notebaley, the model did perform better on understanding references. For instance, given "Question: For what railroad did Stephenson build a locomotive in 1825?" and "Context: Trevithick continued his own experiments using a trio of locomotives, concluding with the Catch Me Who Can in 1808. Only four years later, the successful twin-cylinder locomotive Salamanca by Matthew Murray was used by the edge railed rack and pinion Middleton Railway. In 1825 George Stephenson built the Locomotion for the Stockton and Darlington Railway. This was the first public steam railway in the world and then in 1829, he built The Rocket which was entered in and won the Rainhill Trials. The Liverpool and Manchester Railway opened in 1830 making exclusive use of steam power for both passenger and freight trains.", the baseline give the answer "Prediction: Locomotion for the Stockton and Darlington Railway". However the correct answer is "Answer: Stockton and Darlington". Our model noteably gives a more accurate answer: "Prediction: Stockton and Darlington Railway". Perhaps this indicates that the conditional end pointer is helping the model to distinguish spans of sentences that usually have independent meanings, such as phrases like "<subject> for..."

The model was also better at identify N/A questions. For instance, given the "Question: What term refers to the concrete question to be solved?" and "Context: A computational problem can be viewed as an infinite collection of instances together with a solution for every instance. The input string for a computational problem is referred to as a problem instance, and should not be confused with the problem itself. In computational complexity theory, a problem refers to the abstract question to be solved.", the correct answer is "Answer: N/A". The baseline responds "computational complexity theory" while the model correctly responds "N/A". Perhaps due to the character embeddings, the model is more easily able to form it's unique representations of seldom seen words, which makes it easier for the model to distinguish whether that representation makes sense given the context.

### 6.4 Weaknesses

The model still struggles with questions that phrased slightly differently than the information in the context. For instance, the model still predicted "Prediction: N/A" to the exhaust question mentioned above. This shows that the model still struggles with analogous semantics, such as "brief" and "insufficiently long." This perhaps makes sense because "insufficiently long" does not occur with "brief" often enough to make such connections. Moreover, our additions of character embeddings and conditional end pointer would not help in this case, as there are not out of vocabulary words or tricky start and end positioning. Perhaps further analogy analysis would provide more insight into how the model interprets language.

## 7 Conclusion

Our findings were primarily that the character embedding had the largest impact on performance, and that more feature maps was a critical hyperparameter. Additionally, the answer pointer network was also beneficial to the model. We think that the embedding may have a large impact on models because it determines the information that feeds forward to the rest of the network. Our attention layers did not perform quite as expected, so we ended up using the baseline. Throughout the project, we also followed a structured approach to learning.

While our project attempted to engage in a comparative analysis of various approaches at different stages of the model, we would exercise caution in interpreting our results too directly. While we did conduct an ablation study and controlled certain parameters in our experiments, it is still hard to attribute with certainty that they will always achieve similar performance, given different implementations and different initializations, particularly with the stochastic nature of training.

Ideally, we would have had more capacity to explore the existing literature and review other approaches to the task of question answering. This would have given us a better understanding of the problem domain going into the project, and perhaps a better idea for initial direction, since our project seemed to be mostly determined by our initial plan, even though we adapted along the way. We were in particular disappointed by the poor performance of the coattention and self-attention layers, and are unsure whether this is due to incorrect implementations or some other reason.

Given the constraints in which we did this project, there were many avenues that we did not end up getting to explore. We had been interesting in exploring a beam search approach, similar to the

approach used by Xiong et al. in the Dynamic Coattention Network, as well as the idea of using euclidean distance between words, based on our initial error analysis, but we did not end up having capacity to pursue either of these routes.

Although we did perform error analysis and run various experiments, we did so under some time pressure, and as a result did not get to follow our initial plan. Ideally, we would have gotten to experiment with GRU architectures for encoding and decoding, to experiment more with the relationship between feature maps and hidden size and performance, since these seemed to have the largest individual impact, and to explore what may have possibly gone wrong with our implementations of the attention layers (coattention and self-attention).

## References


[1] Yoon Kim. Convolutional neural networks for sentence classification. 2014.

[2] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. 2018.

[3] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. 2016.

[4] Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task.

[5] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hananneh Hajishirzi. Bi-directional attention flow for machine comprehension. 2018.

[6] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks.

[7] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. R-net: Machine reading comprehension with self-matching networks.


## A Appendix (optional)

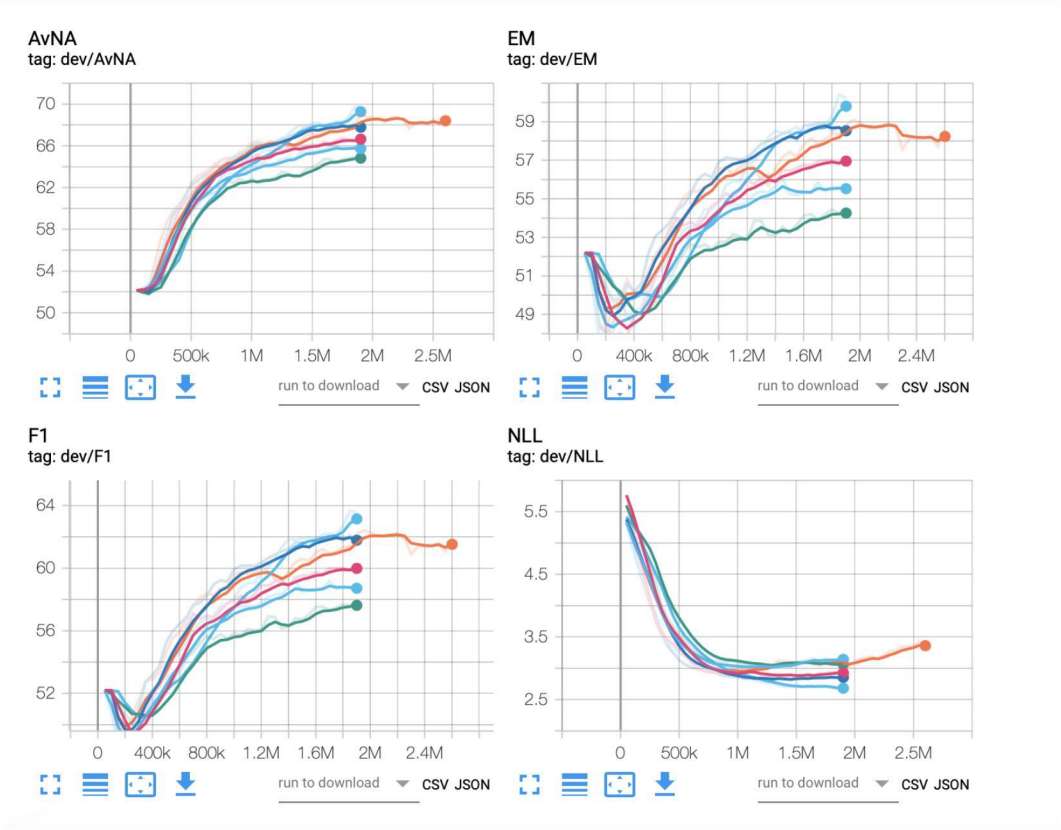


Figure 3: Dev metrics for each of our models. The lower light blue curves are our baseline. The upper light blue curve is our character-embedding + answer pointer network. The dark blue is our character-embedding only. The hot pink is our answer-pointer only network. The teal is our coattention network. The orange is our CharEmb V2 model.

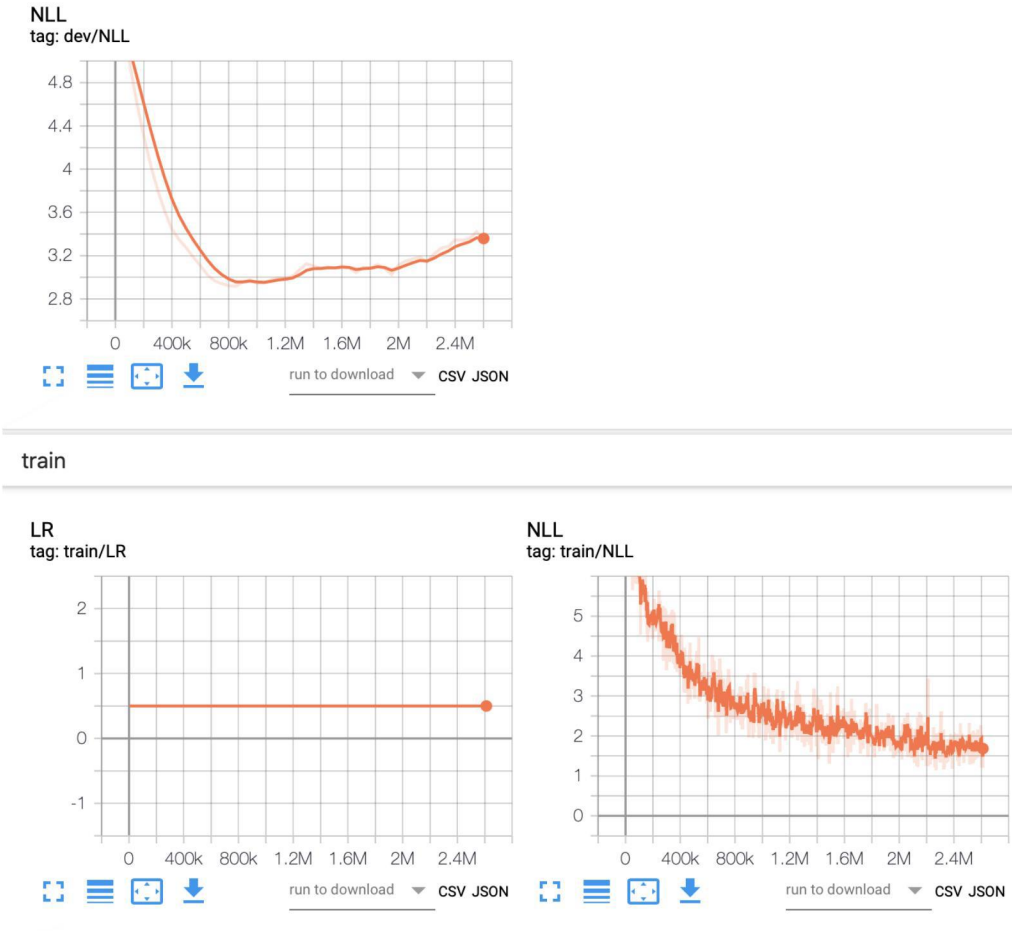


Figure 4: Training and dev losses for CharEmbV2 model