# Building A Robust QA System For IID SQuAD Using Coattention, Self-Attention, and Answer Pointer Network

Stanford CS224N Default Project

**Mostafa Dewidar**
Department of Computer Science
Stanford University
mdewidar@stanford.edu

**George Wang**
Department of Computer Science
Stanford University
gwang4@stanford.edu

**William Cai**
Department of Computer Science
Stanford University
willcai1@stanford.edu

## Abstract

This project explores the problem of question answering on the SQuAD dataset. It evaluates a simple encoder, attention, decoder architecture as a baseline, and shows how some modifications to techniques like coattention, self-attention, answer pointer network, and character embeddings may result in reductions in the reported performances of these techniques. Our best model (resulting from the combination of coattention, character embeddings, and answer pointer network) got EM score of 52.20% and F1 score of 52.20% on the training set.

## 1 Key Information to include

- Mentor: Lucia Zheng
- External Collaborators (if you have any): None
- Sharing project: None

## 2 Introduction

Questions answering is one of the most important areas of research in Natural Language Processing. Since many NLP problems can be formulated in terms of a question answering problem, lots of interest has been spent trying to find good solutions and build robust models for answering questions. As part of these efforts, the SQuAD dataset was released to help benchmark different models and provide ample data for training. The SQuAD dataset is made up of more than 129,000 training data points of questions and contexts pulled from wikipedia. Today, its leaderboard is considered one of the benchmarks of question answering models. In this project, we explore the effects of implementing, as well as modifying some of the techniques used to improve the traditional BiDAF model. The BiDAF model's architecture uses Glove word embeddings as inputs to the mode that are then fed into a simple encoder, decoder architecture with an attention layer applied in the middle. Our project explores the effects of swapping BiDAF attention with coAttention and self-attention, as well as using character embeddings as inputs to the model and conditioning end pointer predictions for the spans of the context that contain the answer on start pointer predictions as well as modifications to these approaches. We find that our modifications to these techniques yield no-to-little improvement on the original implementation of these techniques and can sometimes result in a performance hit.

Our best EM and F1 scores from applying self-attention, character embeddings, and conditioning all at once are at 52.2% each for the training set.

# 3 Related Work

## 3.1 Character-level Embeddings

The baseline model is designed to create embeddings on entire words. However, such a setup does not consider structures within the words, as many English words contain inflections. Thus, [1] also considered embeddings at the character level, which should account for the morphology.

## 3.2 Coattention

Dynamic Coattention Networks (DCNs) [2] improve upon normal attention that uses a single pass mechanism over the question and the query. Instead of looking at the question then the context, linking the important related parts and stopping there, they are more akin to looking at the question, then the context, linking the important parts, then looking at them again. They attend to attention. DCNs do that by first fusing co-dependent representations of the question and the context in order to focus on relevant parts of both. Then a dynamic pointing decoder iterates over po- tential answer spans. This iterative procedure enables the model to recover from initial local maxima corresponding to incorrect answers.

## 3.3 Self-Attention

Self-attention is an alternative attention-based mechanism that is used to improve the question-answering system performance. It has shown to significantly improve the performance of R-NET on the SQuAD and MS-MARCO datasets [3]. The R-NET model uses a self-attention layer to increase its focus on question-relevant evidence in the given context and to enable itself to look over the whole passage to aggregate evidence. The performance boost of the R-NET with the integration of the self-attention layer motivated us to incorporate a self-attention layer into the vanilla BiDAF model to increase its performance.

## 3.4 Answer Pointer Network

The answer pointer network layer is able to achieve the "Conditioning End Prediction on Start Prediction" introduced in the assignment handout. The answer pointer network is able to provide the start token and the end token of the answer by conditioning the probability distribution for the end location on the start location probability distribution [4]. The paper "Machine Comprehension Using Match-LSTM And Answer Pointer" introduces two variants of the answer pointer network layer: The Sequence Model and The Boundary Model. The paper further explains the difference between these two models. The sequence model produces a sequence of answer tokens but these tokens may not be consecutive in the original passage [4]. On the other hand, the boundary model produces on the start token and the end token of the answer, and all the tokens between these two in the original passage are considered to be the answer [4].

# 4 Approach

## 4.1 Character-level Embeddings

The character-level embedding works by first applying a 2D convolutional layer, followed by a MaxPool, before being concatenated to the existing world-level embeddings. Specifically, we follow the approach in [5]. Let $x_i \in \mathbb{R}^k$ be the word vector for word $i$ of a sentence, and define $x_{i:i+j} = x_i \oplus x_{i+1} \oplus \ldots \oplus x_{i+j}$, where $\oplus$ is the concatenation operator. We used the initialized values of $x_{i:i+j}$ provided, which were randomly generated. Then, we feed those values of $x_{i:i+j}$ into a 2-d convolutional layer (Conv2d) with dimensions (1, 5) using default PyTorch parameters for stride, padding, and dilation (1, 0, and 1, respectively). We chose 1 as the first dimension of the kernel size in order to have the appropriate size of the output of the layer, and 5 as the second dimension in order to reflect the width chosen in [1].

Next, we apply MaxPool1d across the max character length, which was 16, in order to reduce the number of dimensions of the output of the 2-d convolutional layer. This involved initializing MaxPool1d using kernel size of 15. We used default PyTorch parameters for padding and dilation being 0 and 1, respectively.

The resulting output was condensed so that it has the same number of dimensions as the word embeddings. The obtained character embeddings were concatenated with the word embeddings, yielding an embedding of double length.

### 4.2 Coattention

As presented in "Dynamic Coattention Networks For Question Answering" [2], the Co-attention layer works by first projecting the encoded, embedded query vectors using a linear pytorch layer, then it obtains an affinity matrix between the projected query and the context using matrix multiplication. It then obtains a context to question attention (C2Q) and a question to context attention (Q2C) by running the affinity matrix through a softmax and summing the affinity results along columns for Q2C, and rows for C2Q to obtain the affinity of each context word to the query in general and for each query word to the context in general. We then combine the C2Q attention distributions ($a^i$) with the Q2C attentions ($b^j$) to attend to the attentions and pass the result through a bidirectional LSTM before feeding it into the decoder network and then obtaining the second-level attention output ($s_i$).

$$s_i = \sum_{j=1}^{M+1} a_j^i b_j \quad \textbf{[4.2.1]}$$

### 4.3 Self-Attention

Our self-attention layer takes in the attention output which contains the context-to-query and query-to-context attention scores as the input (within this subsection, let us denote it as $X$) and will output a distribution of self-attention scores that relates the context to itself.

To implement the self-attention mechanism, we first initialize three learnable weights for "Keys" ($K$), "Query"($Q$), and "Values"($V$) denoted as $K_w$, $Q_w$, and $V_w$.
Then we perform the following arithmetic:

$$K = X \times K_w; Q = X \times Q_w; V = X \times V_w \quad \textbf{[4.3.1]}$$

Then, we can obtain the attention scores for the input $X$ as the following:

$$att = Q \times K^T \quad \textbf{[4.3.2]}$$

After getting the attention scores, we can then apply softmax on the attention scores:

$$SoftMaxAtt = softmax(att) \quad \textbf{[4.3.3]}$$

Finally, we can then multiply the SoftMaxAtt scores with the values we have to produce a weighted values. And then sum up the weighted values to obtain the self-attention scores:

$$SelfAtt = sum(V[:, None] * SoftMaxAtt) \quad \textbf{[4.3.4]}$$

### 4.4 Answer Pointer Network

We chose to use the boundary model as the answer pointer network layer to replace the original BiDAF output layer. This is because the format of the boundary model result[4] is very similar to the format of the output layer result from the original vanilla BiDAF model.

The mathematical model of the answer pointer network is shown as below. Note that $W, b, V, v, c$ are all learnable parameters. And $k$ represents the time step [4]:

$$F_k = \tanh(VH + (Wh_{k-1}^a + b)) \quad \textbf{[4.4.1]}$$

$$B_k = softmax(v^T F_k + c) \quad \textbf{[4.4.2]}$$

$$h_k^a = LSTM(HB_k^T, h_{k-1}^a) \quad \textbf{[4.4.3]}$$

In our model, we only need two time steps to obtain the span of the answer since we only need a start token and an end token to construct the answer. That is we just need to go through $k = 1$ and $k = 2$. $H$, in our model, is the attention output that is a set of representation for all the locations in the given context. Our high level approach to locate the final answer follows section 5.3 in the SQuAD final project hanout [6]. First, the answer pointer network attends to $H$ to produce an attention distribution $B_1$ and a hidden state $h_1^a$ for the start of the answer. Then, we feed the $h_1^a$ to the answer pointer network to produce a new answer pointer hidden state. After that, we then use this new answer pointer hidden state to attend to $H$ to obtain $B_2$.

# 5 Experiments

## 5.1 Data & Evaluation Methods

The data and evaluation metrics for our approaches are the same. The dataset is the SQuAD question answering dataset. The SQuAD dataset contains over 129,000 (context, question, answer) triples. Each context is an excerpt from Wikipedia. The question or query is the question to be answered based on the context. The answer is a span (i.e. excerpt of text) from the context marked with a start pointer pointing to the location of the word at the start of the span in the context array and an end pointer pointing to the location of the end word of the span in the context array. For evaluation we use F1 and EM metrics EM is exact match, a binary measure for whether or not the system outputs and answer that exactly matches the ground truth. F1 is more loose, and is a harmonic mean of the precision and the recall, calculated as:

$$\frac{2 * precision * recall}{precision + recall} \quad \textbf{[5.1.1]}$$

## 5.2 Details

We ran all our experiments with the deafault training parameters of the original BiDAF model. We ran locally on a smaller subset of the training dataset (3 examples for 90 epochs) for sanity tests with a cpu setup. We ran on CoLab using 20,000 examples for 5 epochs. We didn't modify the optimizer, learning rate, or any of the training parameters from their original values.

## 5.3 Results

See figures 1-3 for the plots and corresponding results. Our results for self-attention, combining the former with modified output layers, and combining the former two with character embeddings did not yield different loss as more training was done, suggesting an underlying implementation issue. Our results of character embeddings show promise but did not improve on the baseline. Finally, the coattention method was comparable to the baseline. In the latter two cases, more training can be done to reveal long-term behavior and gain a better understanding of overall behavior.

# 6 Analysis

We observed a lower performance than expected in our experiments. We believe that some of our modifications as well as how we trained the model could explain these performance losses. We outline a couple of the hypothesis we have here:

1- Overfitting: overfitting is when the model only learns to model a training data and learns characteristics that are exclusive to the training dataset, resulting in it failing to generalize to unseen

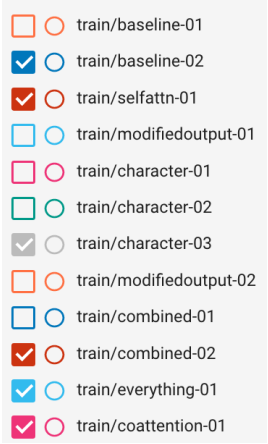


Figure 1: The color-coded key for tensorboard each model tested. baseline-02 is the baseline model; selfattn is the implementation adding only self attention; character-03 adds only the character embedding model; combined-02 combines self attention with the modified output layer; everything-01 combines self attention, modified output layer, and character embeddings together; and coattention-01 is the model that uses coattention.

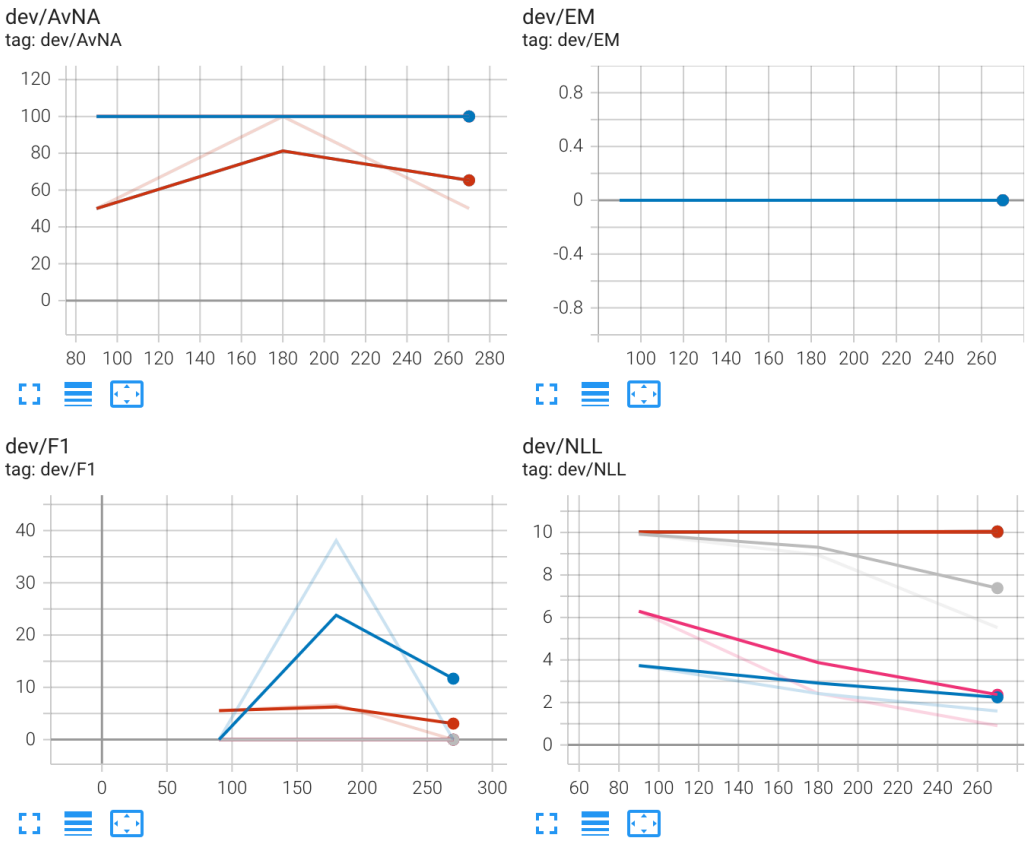


Figure 2: Results on the dev set indicates final F1 scores on the baseline as 11.66 and self-attention as 3.061. The NLL for baseline, character, and coattention were computed as 1.595, 5.529, and 0.9087, respectively. The remaining three methods had a consistent NLL of around 10, suggesting that the models did not learn.

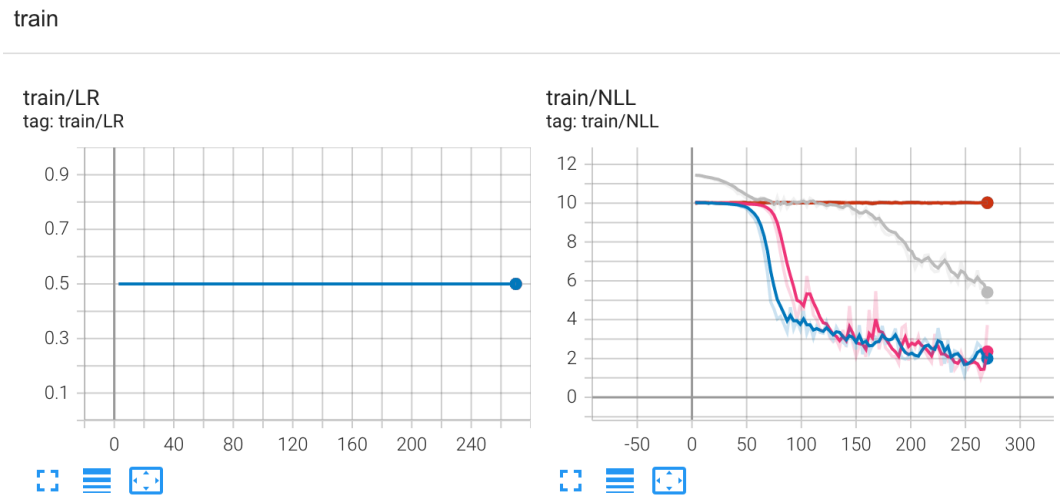


Figure 3: Results of LR and NLL of the different methods. The NLL scores for baseline, character, and coattention are 1.827, 4.796, and 3.72, respectively. The remaining three methods had a consistent NLL of around 10, suggesting that the models did not learn.

data points. We believe we may have led our model to overfit by training for too many epochs / and on a smaller subset of the training dataset. Trying to compensate for hardware limitations on CoLab when the Azure environment stopped working, we reduced the size of our training dataset to about one tenth of the original training dataset and increased the number of training epochs. We found that our loss neared zero at the end of training $2 * 10^{-4}$, however this meant that our model had overfit to the subset of training data that we had provided to it.

2-Effect of removing context and query masks: we tried removing the context and query masking functionality in the original BiDAF model by setting both masks to all ones. Our hypothesis was that a big enough model should be able to distinguish the relationships between the words without the need for masks and that the masks may be removing information that the model could learn from. However, when we trained the model, we found that the model is not able to learn these relationships and yields worse training performance.

## 7 Conclusion

Our work concludes more about our specific training process mistakes and modifications to the techniques rather than the techniques themselves. While we were not able to train the model on the entire dataset due to issues with our azure environment and RAM limitations on CoLab, we were able to segment the dataset into chunks and train different implementations of Coattention, self attention, character embeddings and conditioning. We find that training data performance is not sufficient to make conclusions about the performances of different models and that using a subset of the dataset that is too small won't usually give enough information about the direction of the training. However, We learned how to read and understand research papers in the field of NLP, how to implement these papers, how to understand a big codebase and deal with real-world datasets to build interesting NLP applications. We also learned the importance of good scheduling and planning and the importance of DevOps to the success of any machine learning endeavor as getting azure to cooperate towards the end proved to be the major bottleneck to getting more insightful results on our implementations of these fascinating techniques.

## References


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

[2] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016.

[3] Furu Wei and Ming Zhou. R-net: Machine reading comprehension with self-matching networks, 2017.

[4] Shuohang Wang and Jing Jiang. Machine comprehension using matching-lstm and answer pointer, 2016.

[5] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[6] Cs 224n default final project: Building a qa system (iid squad track), 2022.