Attention Augmentation of BiDAF on SQuAD 2.0

Stanford CS224N Default Project - IID SQuAD track

Nancy Wang
Department of Computer Science
Stanford University
nancyrnw@stanford.edu

Qirui Zhou
Department of Computer Science
Stanford University
qirui@stanford.edu

Abstract

This project aims to build a QA system that gives accurate answers for reading comprehension tasks on the SQuAD 2.0 dataset. On top of the BiDAF baseline model (F1: 59.716, EM: 55.807), we experimented with the combinations of character embeddings, self-attention mechanisms, self-attention and CNN encoder blocks and coattention mechanisms. For each approach, we experimented with either slight modifications of the model structure or different hyperparameters. Then the best approaches are incorporated into our compound model. Our best model combines the baseline with character embeddings, coattention, and self-attention to produce a test accuracy of F1:63.894 (+4.178 on baseline) and EM: 60.168 (+4.361 on baseline). This project shows that the combination of different attention approaches work well together to further improve the performance of the model.

1 Introduction

Question-Answering (QA) has been an essential part of Natural Language Processing. This task is especially challenging for machines since they need to first understand the question and then extract the correct knowledge to answer the question. As one of the hottest research fields in NLP, the improvement in QA algorithms shows how AI is better understanding human language structure, meaning, and reasoning. Powerful real-world applications include question answering on search engines and virtual assistants.

This project focuses on highlighting accurate answers for reading comprehension tasks on the SQuAD 2.0 dataset. While many prior research have been done on approaches like self-attention and co-attention separately, this project verifies that these improving results are reproducible, and further shows that the combination of these approaches work well together to improve the performance more.

2 Related Work

The baseline model employs a technique called bi-directional attention flow (BiDAF) [1] to help obtain a query-aware context representation. One difference between the provided baseline and the BiDAF model is that BiDAF includes a character embedding layer at the beginning while the baseline does not. The BiDAF paper inspired us to implement and experiment with the character embedding layer. Character embedding is very useful when out-of-vocabulary words appear in the corpus because not every word shows up in the word embeddings and character embedding learns the internal structure of words to avoid this problem.

Wang et al. [2] designed an attention-based recurrent network, which inspired us to experiment with self-attention mechanisms on top of the original BiDAF attention layer, which only has context-to-query attention and query-to-context attention. Using self-attention helps the model to aggregate information and evidence from the whole passage, matching the question-aware passage representation to itself and making up the deficiency in the baseline. In addition, we were inspired by the

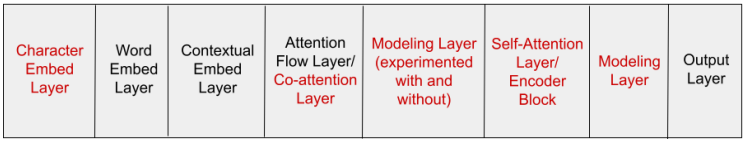
QANet paper [3] to implement stacked encoder blocks with CNN and self-attention mechanisms to replace the single layer of self-attention, utilizing a similar architecture to transformer encoders to provide the advantage of multiple layers of self-attention.

Another type of attention, coattention, was proposed by Xiong et al. [4], which attends to the passage and the question at the same time, and combines both together to calculate the coattention encoding. It inspired us to implement coattention along with self-attention in our model.

3 Approach

3.1 Architecture and Baseline

The architecture of our model contains 8 layers as shown in the figure below (left to right). The layers labeled in black are from the baseline model; the ones in red are our modified layers. Details of the baseline model is in section 4 of the Project Handout [5]. As for our modifications, we added a character embed layer to account for out-of-vocabulary words. We added coattention to the original BiDAF attention flow layer. We also added a self-attention layer on top of the attention flow layer/coattention layer to aggregate information and evidence from the whole passage, along with another modeling layer to further refine the vectors. For the self-attention layer, we experimented with additive self-attention, casual self-attention with multiple heads, or stacked self-attention and CNN encoder blocks. (Our final model uses 1 head casual self-attention). We implemented the layers and models ourselves unless otherwise specified.



3.2 Character Embedding

Our first main approach is to add a character embedding layer at the beginning of the model, and to incorporate it with the word embedding layer to improve the performance. Inspired by the BiDAF paper [1], we created this layer to map each word to a vector space from character-level using convolution neural networks (CNNs). We first used the character-level embeddings to convert each character in the words to vectors. We fed the vectors as input to a CNN and applied one-dimensional max pooling over the output to obtain a fixed-size (which is the hidden size) vector. Then we concatenated the character embeddings with word embeddings together as the input for the next layer instead of only using the word embeddings.

3.3 Self Attention

Our second main approach is adding a self-attention layer on top of the context-to-question attention and question-to-context attention (between baseline modeling and output layer), as shown in the architecture graph. This approach is inspired by Wang et al. [2] to aggregate information and evidence from the whole passage, matching the question-aware passage representation (x_t^P) to itself.

We first calculated the similarity matrix, then passed it through a softmax function to obtain the weights a, and multiplied the weights to original vector values of each word to obtain c. The self-attention output is then concatenated with the original input x and passed through a bi-directional LSTM modeling layer to refine the vectors.

$$h_t^P = \text{BiRNN}(h_{t-1}^P, [x_t^P, c_t])$$

3.3.1 Additive Self Attention

Additive attention was used in the original paper [2], where the attention between two vectors is calculated by: (s_j^t is the similarity between x_t^P and x_j^P)

$$s_j^t = v^T x \tanh(W_x^P x_j^P + W_x^{\tilde{P}} x_t^P) \qquad a_i^t = \exp(s_i^t) / \sum_{j=1}^{n} \exp(s_j^t) \qquad c_t = \sum_{i=1}^{n} a_i^t x_i^P$$

Additive self attention provides additional expressiveness due to the nonlinearity in similarity matrix calculation. However, this approach has very expensive computation and storage costs. Thus, we also experimented with causal self-attention.

3.3.2 Causal Self Attention

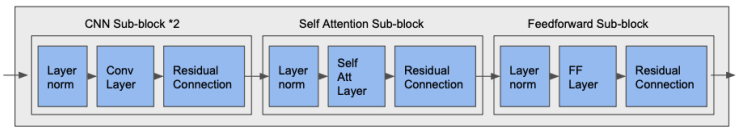
We experimented with single and multi-head causal self-attention which was inspired by CS224N lectures. The attention is calculated by: (C_i is the output of one attention head. Q, K, V are Key, Query, Value matrix used for causal self attention. d is the total dimensionality. h is the number of heads.)

$$S_i = \frac{(XQ_i)(XK_i)^T}{\sqrt{d/h}} \qquad A_i = \text{softmax}(S_i) \qquad C_i = A_i\,(XV_i)$$

Causal self attention allows interaction between input vectors directly because of the dot product, which provides more localized attention. It also reduces training time and storage needed. We referenced the attention file in assignment 5 when coding our multi-head causal self-attention.

3.4 Self Attention and CNN Encoder Block

On top of 3.3, we wanted to experiment with more layers of self-attention in our model. We did not simply stack more self-attention layers on each other without adding non-linear layers since that would achieve similar results as increased-head causal self-attention. Instead, we were inspired by the encoder blocks of transformers [6] and the QANet paper [3], which lead us to create our own encoder block structure, as shown below:



We first passed the input vectors through the depthwise separable convolutional sub-block two times to aggregate the weighted information from the input vectors. Then it is passed through the self-attention sub-block, which uses the causal self attention we implemented in 3.3.2. Finally it passes through a feedforward layer that introduces non-linearity. In each sub-block, a layernorm and a connection layer are added to allow the model to train faster. We used the depthwise separable convolution function from [7], but implemented the encoder blocks ourselves. With multiple blocks stacked together, we hypothesize that it achieves the advantage of using multiple self-attention layers.

3.5 Coattention

We first added a sentinel vector at the end of both passage (D) and question matrix (Q). This allows words to not attend to any word in the input if the words do not match. We also added a projection layer for the question encoding to introduce non-linearity ($Q = \tanh(WQ + b)$). Then we calculated the affinity matrix ($L = D^T Q$). Further on, we normalized the affinity matrix to calculate attention weights ($A^Q = \text{softmax}(L)$, $A^D = \text{softmax}(L^T)$), and computed the summaries of the passage($C^Q = DA^Q$) and question (QA^D). Next we calculated the summaries of the previous attention contexts taking each word of the passage into consideration and combining with the summary of the question.

$$[QA^D, C^Q A^D] \Rightarrow C^D = [Q; C^Q]A^D$$

Viewing them together (C^D), we have a co-dependent representation of the question and the passage, which is the coattention context, and needs to be passed through a bidirectional LSTM to build coattention encoding.

4 Experiments

4.1 Data and Evaluation Method

We are using the official SQuAD 2.0 dataset containing 129,941 training examples, 6,078 eval example, and 5,915 test examples. We will be evaluating our results from the F1 and EM (exact match) metrics as mentioned in the Default Project Handout [5].

4.2 Experiment Details

For the training process of the models, we used a learning rate of 0.5, dropout probability of 0.2, seed of 224, and trained the model for 30 epochs, unless otherwise specified.

Similar to the word embedding, we added the character embedding by loading in the pretrained character vectors. We added a 1-dimensional CNN layer with the input size of the character embedding size, the output of hidden_size, and the kernel size of 5. A 1-dimensional max pooling is applied on the CNN output with a kernel size of the innermost dimension of the CNN output, followed by a dropout layer. In the character embeddings experiment we doubled the hidden_size parameter (to 200) since we wanted to keep the original input-to-hidden-layer size ratio. We experimented with different learning rates (0.1, 0.3, 0.5, 0.7, 0.9) and dropout rates (0.1, 0.2, 0.3) as our hyperparameter tuning. (Please see results section.) The training time for each epoch increased to about 40 minutes on the Azure virtual machine.

For self-attention, we experimented with additive self-attention and causal self-attention with 1,2,4 heads. We also experimented with the dropout residual (see results section). We followed the dimensions and equations in 3.3 for the implementation, and initialized all parameters randomly. We kept the original hidden size of 100. Since additive self-attention is computationally expensive, the training time increased to around 60 minute per epoch. The training time for causal self-attention is around 35 minutes per epoch.

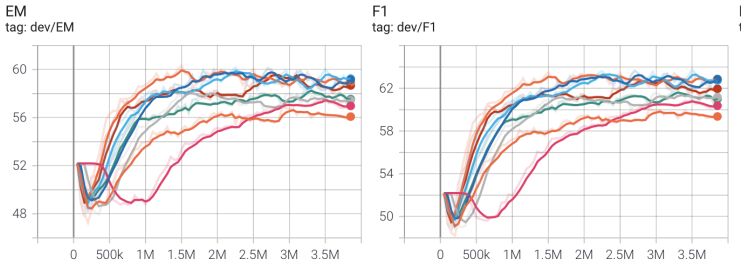
For the self-attention and CNN encoder block layer, we used 2 CNN layers, each with kernel size of 5 and hidden size of 100. We stacked 3 encoder blocks in our model, sharing weights between them. The training time is around 40 minutes per epoch.

For the coattention mechanism, our implementation followed the equation details we explained in 3.5. We used a hidden size of 100. We otherwize used the default parameters mentioned above. The training time is around 30 minutes per epoch.

4.3 Experiment Results

Please see the following tables for our experiment results. We did 4 sets of experiments: 1) baseline + character embedding 2) baseline + self-attention 3) baseline + self-attention CNN encoder block 4) Compound models combining baseline + character embedding + coattention + self-attention. For each set, we experimented with either varying hyperparameters or slightly changing the model structure. F1 and EM scores are dev eval results unless otherwise specified (for the final model).

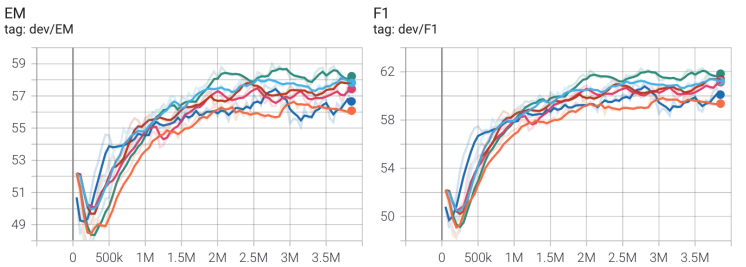
Ref Number	Model	Learning Rate	Dropout Rate	F1	EM
#0	Baseline	0.5	0.2	60.06	56.86
#1	Baseline + Char Emb	0.1	0.2	60.92	57.65
#2	Baseline + Char Emb	0.3	0.2	61.55	58.02
#3	Baseline + Char Emb	0.5	0.2	63.66	60.12
#4	Baseline + Char Emb	0.7	0.1	63.47	60.34
#5	Baseline + Char Emb	0.7	0.2	63.78	60.04
#6	Baseline + Char Emb	0.7	0.3	61.81	58.51
#7	Baseline + Char Emb	0.9	0.2	63.06	59.82



Plots for models #0 - #7 in the above character embedding results table

We did 6 character embedding experiments by changing the learning rate and dropout rate. The best result is given by model #5 (learning rate=0.7 and dropout rate=0.2), yielding a dev eval result of F1:63.78 and EM:60.04. The second best model #3 (learning rate=0.5 and dropout rate=0.2) has very similar F1 and EM scores as #7. Since model #7 was a later experiment we did, we used model #3 as our character embedding model in the compound models.

Ref Number	Model	# of Heads	Add Residual Dropout	F1	EM
#0	Baseline	NA	NA	60.06	56.86
#8	Baseline + Additive Self Attention	NA	NA	61.14	57.62
#9	Baseline + Causal Self Attention	4	T	61.42	57.69
#10	Baseline + Causal Self Attention	2	T	61.66	58.09
#11	Baseline + Causal Self Attention	1	T	61.71	58.48
#12	Baseline + Causal Self Attention	1	F	62.30	58.85



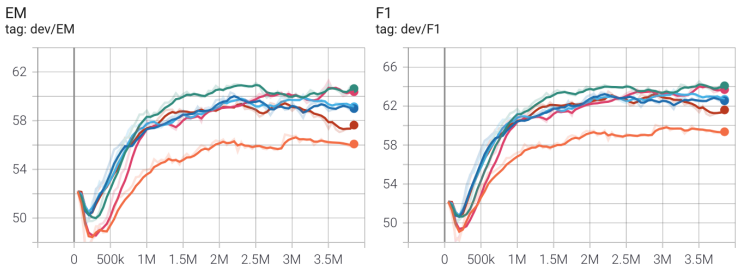
Plots for models #0 and #8 - #12 in the above self attention results table

For the baseline + self-attention experiments, results show that additive self-attention performs worse than causal self-attention. 1 head causal self-attention performs better than 2 head and 4 head causal self-attention. And the model without an addition residual dropout performs better. The best model in this section is the 1 head causal self-attention without an additional residual dropout layer (#12), yielding a dev eval result of F1: 62.30, EM: 58.85. We used this 1 head causal self-attention model for our compound models.

Ref Number	Model	F1	EM
#0	Baseline	60.06	56.86
#13	Baseline + Encoder Block (3 blocks)	60.25	56.68

The result of the encoder block is not as high as we expected, yielding only a slight improvement with a dev evaluation of F1: 60.25, EM: 56.68. We will analyze the reasons to this in the next section. This model did not make it into our final compound model.

Ref Number	Model	Hidden Size	Add Residual Dropout	Modeling Layer	F1	EM
#0	Baseline	100	NA	NA	60.06	56.86
#14	Baseline + Char Emb + Self Attention	200	T	T	63.41	60.33
#15	Baseline + Char Emb + Self Attention	200	F	T	63.50	59.65
#16	Baseline + Char Emb + Self Attention	100	F	T	63.67	60.14
#17	Baseline + Char Emb + Self Attention	100	F	F	64.46	61.28
#18	Baseline + Char Emb + Coattention + Self Attention	100	F	T	64.55	60.96



Plots for models #0 and #14 - #18 in the above compound models results table

In our compound model experiments, we first experimented with combining baseline+char embedding + self-attention, then experimented with combining all of baseline+char embedding + self-attention + coattention. We have concluded that hidden size=100 performs better than hidden size=200, and an additional residual dropout layer should not be added. Removing the modeling layer between the context-to-question/question-to-context attention and self-attention layer also improves performance.

Ref Number	Model	F1 (Test Result)	EM (Test Result)
#0	Baseline	59.716	55.807
#17	Baseline + Char Emb + Self Attention	62.819 (+3.103)	59.394 (+3.584)
#18	Baseline + Char Emb + Coattention + Self Attention	63.894 (+4.178)	60.168 (+4.361)

We choose compound model #17 and #18 to evaluate on the test set. Compared to the baseline model, combining baseline+char embedding + self-attention + coattention (#18) yields the highest scores of F1: 63.894 (+4.178) and EM: 60.168 (+4.361). This is the final best model of our paper.

5 Analysis

5.1 Model Analysis

5.1.1 Character Embeddings

When we implemented the character embedding layer for our model, we chose to use 200 as our hidden size (which is twice of the original hidden size) for all layers after it, because the input size is doubled for those layers and we want to keep the original ratio between the input size and the hidden size. The addition of character embedding layer significantly improved the performance and F1 and EM scores. Both scores go up about 3 points. This performance improvement is within our expectation. It indicates that a character-level embedding layer is important and useful in our scenario because when it meets out-of-vocabulary words, it could still learn the internal structure of those

words, while only using word embeddings could not. Our experiment results verifies our hypothesis about character embedding.

5.1.2 Self Attention

Causal self attention worked well as we expected. However, additive self attention did not. We were surprised at first since additive self attention provides additional nonlinearity expressiveness in its similarity matrix calculation. After closer analysis, we realized that additive self attention lacks the direct interaction between input that causal self attention has through the dot product of two vectors in its similarity matrix calculation, which provides more localized attention. Due to this reason, causal self attention out performs additive self attention.

5.1.3 Self Attention and CNN Encoder Block

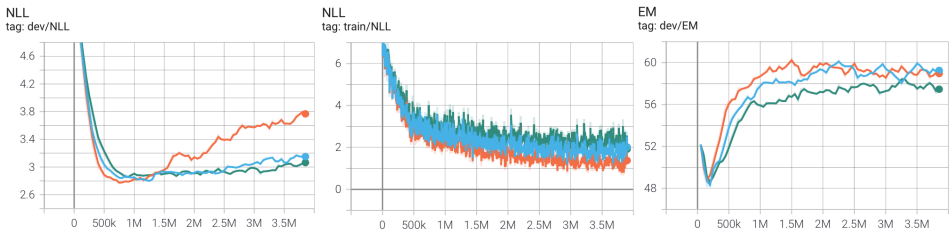
The Self attention and CNN Encoder Block did not work as well as expected. The accuracy improvement is very limited. We think there are mainly two reasons. Most importantly, compared to the QANet paper [3], we added an RNN modeling layer after the BiDAF attention before the encoder blocks in our architecture, but the original paper added the encoder blocks directly after the context-query attention. This could be a problem since the encoder block relies on positional information to perform CNN and self-attention. However passing the BiDAF attention through an LSTM aggregates information in the different positions and loses positional information encoding. Another reason is that the paper [3] used the intermediate output of different encoder blocks for the final position prediction, but we only relied on the output of the final layer of the encoder block. A next step direction is to experiment with encoder blocks that do not have a modeling layer after the BiDAF attention, and use intermediate block outputs for answer predictions.

5.1.4 Coattention

The coattention mechanism has some but not outstanding performance improvement. It increased the test F1 and EM scores by around 1 point each compared to the model that did not use coattention. This result is slightly unexpected since we were expecting larger improvements. After closely analyzing the coattention and the original attention flow layer in the baseline model, we found out that they both share some common computation process, for example, the way context is attended to the question (dot product, softmax used). This overlap shows that they may extract similar information from input vectors, which explains why coattention did not improve the baseline (which already uses BiDAF attention) by a large margin.

5.2 Training Evaluation

During our experiments with character embedding, we found that among the three models with a learning rate of 0.7, a smaller dropout rate would cause slight overfitting. The graph below shows the negative log-likelihood dev loss for the three models, where the orange line represents the model with a dropout rate of 0.1. It stopped minimizing and began rising in its midway, while its training loss kept on decreasing.



We also found that the model with a smaller learning rate has dev F1 and EM scores that fluctuate less than the one with a larger learning rate. Please see Appendix A for the detailed graph plotting.

5.3 Final Model Text Qualitative Evaluation

For error analysis, we are comparing the baseline model with our final model (#18).

Generally, our model #18 was able to predict long answers better, as shown in the below example, while the baseline model could not locate the correct answer and predicted N/A.

- **Question:** What was AUSTPAC
- **Context:** AUSTPAC was an Australian public X.25 network operated by Telstra. Started by Telecom Australia in the early 1980s, AUSTPAC was Australia's first public packet-switched data network, supporting applications such as on-line betting, financial applications — the Australian Tax Office made use of AUSTPAC — and remote terminal access to academic institutions, who maintained their connections to AUSTPAC up until the mid-late 1990s in some cases. Access can be via a dial-up terminal to a PAD, or, by linking a permanent X.25 node to the network.[citation needed]
- **Answer:** AUSTPAC was an Australian public X.25 network operated by Telstra
- **Prediction:** an Australian public X.25 network operated by Telstra

We found two general errors in our final model: 1) when there are synonyms in the question and the correct answer, the model is not able to identify the synonyms and thus unable find the answer. In the example below, the phrase "insufficiently long" is a synonym for "too brief", which should be an indication that the answer is in or around this phrase, but the model failed to find the answer.

- **Question:** What can the exhaust steam not fully do when the exhaust event is insufficiently long?
- **Context:** The simplest valve gears give events of fixed length during the engine cycle and often make the engine rotate in only one direction. Most however have a reversing mechanism which additionally can provide means for saving steam as speed and momentum are gained by gradually "shortening the cutoff" or rather, shortening the admission event; this in turn proportionately lengthens the expansion period. 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 ("kick back").[citation needed]
- **Answer:** evacuate the cylinder
- **Prediction:** N/A

2) We also found out that our model #18 often predict slightly incorrectly when the answer involves numbers. Especially for propositional words around year/time.

- **Question:** When had the Brotherhood renounced violence as a means of achieving its goals?
- **Context:** While Qutb's ideas became increasingly radical during his imprisonment prior to his execution in 1966, the leadership of the Brotherhood, led by Hasan al-Hudaybi, remained moderate and interested in political negotiation and activism. Fringe or splinter movements inspired by the final writings of Qutb in the mid-1960s (particularly the manifesto Milestones, a.k.a. Ma'alim fi-l-Tariq) did, however, develop and they pursued a more radical direction. By the 1970s, the Brotherhood had renounced violence as a means of achieving its goals.
- **Answer:** By the 1970s
- **Prediction:** 1970s

6 Conclusion

In this project, we experimented with various character embeddings, self-attention mechanisms, self-attention and CNN encoder blocks and coattention mechanisms on top of the BiDAF baseline model. We achieved a final **F1 score of 63.894 and EM score of 60.168** in our baseline BiDAF + character embeddings + coattention + 1 head causal self-attention model (#18), which improved the baseline F1 score by +4.178 and the baseline EM score by +4.361.

In previous papers, there are results that show how each singe approach improves the F1 and EM scores. In our project, we verified that these improving results are reproducible, and **further showed that the combination of these approaches work well together** to improve the performance more.

A limitation of this model is that it concatenates vectors at each position instead of allowing the model to learn a weighed average. For example, we concatenated the coattention output with the self-attention output instead of allowing a weighting mechanism for the model to learn the importance of each. This also means that the vectors at each position becomes longer as more attention mechanisms are used, thus increasing computation cost. One future work is to add weights for the model to learn the importance of each mechanism and combine them in the best way.

Another future work is to provide a voting mechanism like ensemble learning to combine the different attention mechanisms. Currently we are stacking the different attention mechanisms together in one model, but it may be beneficial to create a model with each mechanism and ensemble them to get the final output.

References

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

[2] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 189–198, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[3] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension, 2018.

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

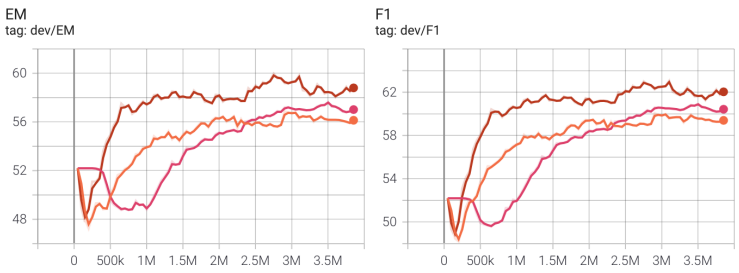
[5] CS 224N Teaching Staff. Cs 224n default final project: Building a qa system (iid squad track), 2021.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[7] heliumsea. Qanet-pytorch. `https://github.com/heliumsea/QANet-pytorch/blob/02bd338262732c6a01fc91081fe47c36bbcd2e04/models.py#L39`, 2019.

A Appendix

Affect of learning rate on dev EM and F1 in character embedding experiments:



From the graph above, we can see that the pink line, which corresponds to the model with a learning rate of 0.1, is much more smooth than the red line with a learning rate of 0.9.