Question Answering with Co-attention and Transformer

Stanford CS224N Default Project

Xiran Shi, Xintong Yu
ICME
Stanford University
xiranshi@stanford.edu, xy10@stanford.edu

Abstract

In this project, we investigated the performance improvements of incorporating coattention layer, QNet, and RNet to the basic Bidirectional Attention Flow model for QA systems. We constructed the two models with one used QANet and the other implements char embedding and self attention, trained on on SQuAD 2.0 dataset, and evaluated the improvements based on the Exact Match and F1 scores. The two model respectively improve the EM from **57.39** to **64.49** (QANet), **62.03** (CharEmbed + Self Match), and F1 from **60.64** to **69.62** and **65.53**. With the evaluation results, we further analyzed the advantage and limitations of each approach qualitatively.

1 Key Information to include

- Mentor: Daniel Do

2 Introduction

Since the release of Stanford Question Answering Dataset (SQuAD), rapid progresses in reading comprehension style question answering have been made. The current state of art model can generate 90% of the answers exactly. [1]. Though many of the recent progresses utilized BERT, in this paper, we focused on using transformer and additional attention layer to improve the performance of the baseline model.

First, we implemented the co-attention layer, which captures the interaction between context and question attention. The co-attention layer failed to improve the model performance but generated very similar results as the baseline model. We will analyze the failure reasons later in the report.

Second, we implemented QA-Net which uses convolutions to capture the local structure of the context and self-attention mechanism to model the global interactions between text. We were able to improve the model performance significantly compared to the baseline model. The QA-Net is not built on the top of baseline model but replaces the RNN in baseline with convolution network.

Finally, we implemented the self-matching layer and pointer network as described in RNet to the baseline model. The self matching mechanism helps refine the attention representation by matching the passage against itself, which effectively encodes information from the whole passage. The pointer network, which is output layer of RNET, predicts a pointer to the start and end points of the answer spans.

3 Related Work

Over the past few years, significant progress has been made in the question answer system, with which the model generally employ two components: (1) a recurrent model that takes sequential inputs, and (2) attention component. A successful employment of the two components is Bidirectional Attention Flow (BiDAF) by Seo, Kembhavi, Hajishirzi 2016 [2], which is also the baseline model of the project. The paper proposed a multi-layer hierarchical structures to give a bidirectional attention flow network. This enables a query-aware context representation. The Dynamic Coattention Network (DCN) by Zhong, Xiong, Socher, 2016, fuses the co-dependent representations of questions and context and then predicts the relevant answer span, which enable the model to recover from local maxima corresponding to incorrect answers. The Microsoft NLP group, 2017, introduced R-NET. R-NET includes a self-matching attention mechanism to refine the representation by matching the passage against itself, which effectively encodes information from the whole passage. Pointer networks are employed to locate the positions of answers from the passage. All mentioned models are primarily based on recurrent neural networks, while Yu & Dohan 2018, proposed a new QA architecture that does not reply on recurrent networks. It employs convolution as local interaction model and self-attention as global interaction models. The method is faster to train because of its feed-forward nature, and the paper assumes it is 3x to 13x faster to train than BiDAF. The paper also achieves a best published F1 score of 84.6.

In the related work, we only name a few papers relevant to our project but those are far beyond exhaustive. By far, the highest F1 and EM score on the leaderboard has reached 90.87 and 93.18. Many of those models uses ensemble models and utilize BERT.

4 Approach

We implemented: **Character-level Embeddings**, **QANet** [3], **Co-attention Layer** [4], and **RNet** [5]. We wrote the code ourselves but did use online github code for reference.

We used the BiDAF model provided as our baseline model. The goal is to out perform the EM and F1 (evaluation metric) for the provided model.

4.1 Character-level embeddings

For Character-level embeddings, we first imported the pretrained GloVe character embeddings, representing each character with a 60-dimensional vector. We then fed them into a 1D Convolutional Neural Networks (CNN) layer. The resulting vector is then max-pooled along the dimension of kernel applications which then produces character embedding vectors for the entire word. These vectors are then concatenated with the projected pretrained GloVe word embedding vectors to form vectors, which are then fed into the Highway Network.

4.2 Co-attention Layer

The co-attention layer performs a series of computation as described. This process computes the context to question and question to context attention, concatenates the results together and outputs that as a co-attention layer.

The co-attention layer first defines the Affine matrix L that composed the information from both context and question. $L = D^T Q$, while D, Q are the document and question representation.
It then produces attention weights A^Q, A^D for the question and the document: $A^Q = softmax(L)$, $A^Q = softmax(L^T)$

The attention context, which is the attention of document in light of each question, is computed: $C^D = DA^Q$ Similarly, attention to question in light of each document word is calculated: $C^Q = QA^D$.
We then compute $C^Q A^D$which is the attention summary of the question in the light of the previous attention layer.
The final step is to apply a bidirectional LSTM layer as fusing the current information to the co-attention layer.

$$u_t = Bi - LSTM([d_t; c_t^D])$$

$U = [u1, u2, ...,]$ is the output of the co-attention layer.

Since the bi-LSTM architecture is already implemented in the `class RNNEncoder`, the output is $[d_t; c_t^D]$ and will be used as input to `BiDAFOutput` layer.

4.3 RNet

We implemented the self-matching layer and the pointer network proposed by RNet 2017 [5]. We used the code [6] as reference for certain ambiguity in the paper.

4.3.1 Self Matching

The self matching layer enables a dynamic collection process of answer evidence from the passage. It encodes the relevant evidence in current passage word to the whole passage.
$\{v_t^P\}_{t=1}^n$ is the attention vector, and we computes the followings:

$$s_j^t = v^T tanh(W_v^P v_j^p + W_v^{\widetilde{P}} v_t^P)$$
$$a_i^t = softmax(s_i^t)$$
$$c_t = \sum_{i=1}^n a_i^t v_i^P$$
$$h_t^P = BiRNN(h_{t-1}^P, [v_t^P, c_t])$$

The above in sequence computes the connection between attention of word j and word t as s_j^t. It then obtains the distribution of s through computing $a = softmax(s)$, and finally computes c by multiplying distribution of s and the attention vector. With trainable parameter W_v, s can give a representation of two passage words relation.

4.3.2 Pointer Network

The RNet adopts the same approach of Wang Jiang 2016 [7] and use the pointer newtwork Vinyals et al., 2015[8] to predict the start and the end position in the answer. The pointer network uses an attention-pooling over the question representation to generate the initial hidden vector for the pointer network and select the start and end position.

The h_{t-1}^a represents the last hidden state of the pointer network.

$$s_j^t = v^T tanh(W_v^P h_j^P + W_h^a h_{t-1}^a)$$
$$a_i^t = softmax(s_i^t)$$
$$p^t = argmax(a_1^t, \ldots, a_n^t)$$
$$c_t = \sum_{i=1}^n a_i^t v_i^P$$
$$h_t^P = BiRNN(h_{t-1}^P, [h_{t-1}^a, c_t])$$

a_i^t is the predicted distribution of the start position p_t. With a has been calculated, we use that to further compute the end position of answer by going though a bidirectional GRU. By repeating the procedure, we obtion the predicted distribution of the end pointer.

To predict h_{t-1}^a, we follow the procedure below. The h_{t-1}^a is initialzed with quesiton vector r^Q, which is an attention pooling vecotr of question based on a parameter V_r^Q

$$s_j^t = v^T tanh(W_u^Q u_j^P + W_v^Q V_v^Q)$$
$$a_i^t = softmax(s_i^t)$$
$$r^Q = \sum_{i=1}^n a_i u_i^Q$$

4.4 QANet

The model architecture of QANet is similar to BiDAF to some extent, but there are several major differences.

QANet contains the following five layers: input embedding layer, embedding encoder layer, context-query attention layer, model encoder layer and output layer.

The input embedding layer makes use of character-level embeddings, the only difference is that instead of pretrained GloVe character embeddings, 60-dimensional trainable vectors are used, which is then fed into 1D CNN layer with 200 output channels, providing 200-dimensional character embeddings.

In the embedding encoder layer, first, the sinusoidal position representations are obtained using the following functions:

$$PE_(pos, 2i) = sin(pos/10000^{2i}/d_{model})$$
$$PE_(pos, 2i+1) = cos(pos/10000^{2i}/d_{model})$$

We used the implementation from PyTorch [9]. This is summed with the output from the input embedding layer to incorporate positional information. Second, a stack of building blocks are built. Each building block contains 4 depth-wise separable convolution layers. After that comes the self-attention layer with multi-head attention mechanism. This part of the implementation is adapted from the stencil code of Assignment 5. The number of heads is 8. Finally, a 1D CNN layer with kernel size of 1 is applied. Each of these operations are preceded by layer normalization, and each combination of layer normalization and operation is placed within a residual block with dropout (adapted from [10]), which can we represented with $f(layernorm(x)) + x$. We directly used the original stencil code for the implementation of context-query attention layer.

The model encoder layer consists of a linear feed-forward layer that converts the input into dimension of 128, and a total of 7 stacked encoder blocks, each equivalent to the embedding encoder layer. The input to model encoder layer goes through the stacked blocks a total of 3 times, giving outputs $ME1$, $ME2$ and $ME3$. All three iterations share weights. Finally the output layer produces probabilities of the starting position through concatenating ME1 and ME2 and then feeding it into a linear layer followed by softmax, and probabilities ending position calculated in a similar way but using concatenation of ME1 and ME3 instead. The operations are represented with $p_1 = softmax(W_1[ME1; ME2]), p_2 = softmax(W_2[ME1; ME3]),$.

5 Experiments

5.1 Data

The dataset is the SQuAD 2.0 dataset as given. SQuAD 2.0 dataset consists of 100,000+ questions and answers. The questions are posed based on Wikipedia acticles, and the answers can be found from the corresponding article. The QA system is trained based on the triplet of given corpus, questions, and corresponding answers.

5.2 Evaluation method

To evaluate the accuracy of the QA system, we adopt two metrics: **Exact Match**, (EM) score and **F1** score. We average EM and F1 as the ultimate metric.

5.3 Experimental details

5.3.1 Character-level Embedding

For character-level embeddings, we used one 1D CNN layer with kernel size of 5, output channels of100 and stride of 1, and one max-pooling operation. For all other parameters, we used their default values from the provided code.

5.3.2 QANet

After implementing character-level embedding, we first implemented the embedding encoder layer only and replaced the RNN encoder layer in the BiDAF model. The implementation (implementation 1) included depth-wise separable convolution layers, self attention layer and residual block structure with dropout ratio of 0.2 adapted from [10]. We did not include positional encoder at this point. We used default batch size of 64, and the training time was approximately 40 minutes per epoch. After 30 epochs, the EM score and F1 score on the dev set were 59.4 and 63.3.

After that, we implemented the model encoder layer (implementation 2), and the EM and F1 score at 1 million iteration was only 55.81 and 57.68 (that of char-level embedding was 57.65 and 60.69).

Therefore we added positional encoder to see the effects (implementation 3), and the EM and F1 score were basically on par with the previous implementation. At this point, we reduced batch size to 16 because of memory issue, and the training time for each epoch was approximately 1.5 hours.

We then replaced the pre-trained character embeddings with trainable embedding vectors, and removed dropout from the residual block structure (implementation 4). The results were were not good, with F1 score of only 53.45 at epoch 6 (that of char-level embedding was 60.28). Since we identified overfitting, we decided to reverse back to using pre-trained character embeddings with dropout in the residual block and reduced the number of heads for multi-head attention to 4 (implementation 5). The results were satisfactory with EM and F1 score of 65.32 and 68.33 at the highest. Up to this point, we did not add masking for the padding tokens in the multi-head attention layer. Therefore we added masking and increased the number of heads back to 8 to boost performance(implementation 6). This is our best-performing implementation with EM and F1 score of 64.49 and 69.62 at the highest.

5.3.3 RNet

We built Self Attention with output from the `BiDAF Attention` layer. Rather than computing $s_j^t = v^T tanh(W_v^P v_j^p + W_v^{\widetilde{P}} v_t^P)$ for each pair, we flattened the input attention and compute s_j with the flatted attention. The original RNet utilized a gate attention-based recurrent networks for the final output $[v_t^P, c_t]$ to control the input of RNN. The paper did not include enough details for the gate, and we used `Highway Encoder` which served a similar goal. This step turned out to be crucial, and improved both the EM and F1 by roughly 1%.

A further modification is to replace the LSTM in `BiDAF Output` with GRU. Such modification reduces the per epoch time by about 10 minutes. The yields so far the best RNN performance: EM and F1 are 62.06 and 65.53 respectively.

We implemented the Pointer Network afterwards. We used `GRUCell` for the RNN, and included 4 trainable weights. In one layer, we in sequence applied linear transform, dropout, gru, and softmax. We included 3 layers here. This implementation causes vanishing gradients. The gradients decays to zero rapidly.

5.3.4 Coattention Layer

The implementation of co-attention layer is relatively straightforward as the formula above described. The input the is `BiDAF Attention` output and we applied dropout and masked softmax to prevent overfitting. Inclusion of coattention layer to the baseline did not improve the performance. Even worse, after incorporation Character Embedding, the performance downgraded. Failure reasons will be analyzed.

5.4 Model Configuration

One distinct difference between QAnet and the RNet is that the RNet, same as the baseline and many other QA system, uses RNN for modelling component but the QANet uses convolution instead. Thus, it is not feasible to merge the architecture of QANet and RNet. Instead, we train each model separately and compare their performance and efficiency.

5.4.1 Hyper parameter Tuning

Unfortunately, due to the credit budget, we were not able to perform many fine tuning on the hyper parameter. We tested a very small set of hyper parameter, and do not run for 30 epochs. We left the learning rate unchanged, but modified the hidden size to 128 for QANet. We also modified the number of hidden layers to 3 for the self match output layer.

5.4.2 Training Time

We trained for 10-30 epochs for each model, which take from 10 hours to 40 hours. We would interrupt the process if there is clear over-fitting. For instance, when the Dev Error increases substantially and is much larger than the training error.

5.5 Results

5.5.1 Evaluation Metrics

The followings are results after 30 epochs.

	Baseline	CharEmbed	CharEmbed+SelfMatch	QANet	Test Leaderboard (QANet)
EM	57.39	60.59	62.06	64.49	62.502
F1	60.64	64.17	65.53	69.62	65.953
Avg(EM, F1)	59.01	62.38	63.80	67.06	64.228

- QANet improves EM and F1 scores to 64.49 and 69.62. The results are satisfying.
- Self Match + Character Embedding. The result is slightly lower than expected. The RNet achieves EM and F1 of around 75, 79. However, we failed to utilize the Pointer Network and have some modifications in the model structure, so the result is acceptable.

5.5.2 Efficiency

We compared the training time of two models configurations. The QANet is relatively slow to train.

	Baseline	QANet	CharEmbed + Self Matching
time/epoch (min)	22	100	65

5.5.3 Visualization

Visualizations are in appendix: QANet fig 1, SelfAtt + Char Embed fig 2.

6 Analysis

6.1 QANet

We can identify multiple interesting trends by examining the EM, F1 and loss curves (in Appendix).

In the results for implementation 4, we see that at the very beginning, the F1 and EM scores are much higher and the loss is much lower for the dev set compared to all other implementations. However, the scores stop to increase and fluctuate violently very soon, and the loss decreases much slower than other implementations. This indicates that simply using $f(layernorm(x)) + x$ for residual block implementation without dropout, together with trainable character embeddings and 8 heads achieves high performance early on with lots of learnable model parameters, but lacks momentum for further improvements. It is possible, though, that if it was allowed long enough training time, performance might boost.

When comparing the performances of implementation 1 and character-level embedding, we can see that although character-level embedding achieves higher scores, the F1 and EM curves plateau and even start going downwards after 3 million iterations, while the curves for implementation 1 go up relatively steadily. Comparing implementation 3 with character-level embedding, we see that although the scores are much lower, the loss curve plateaus and does not go back up. It seems character-embedding has a more serious problem of overfitting.

There are multiple interesting observations for implementation 5. Examining the loss curve for development set, we see that up to 1.6 million iterations, which is about half way into training, compared with all other implementations, the loss first decrease most rapidly and plateaus the earliest, showing signs of overfitting. This can also be observed in the EM and F1 curve where the scores first increase rapidly and plateaus early. However after that there is a clear drop in the loss curve from 2.8 to 2.6 from 1.6 million iterations to 1.9 million iterations where the curve plateaus again and increases, again showing signs of overfitting. The drop in loss can also be observed in the EM and F1 curves in the form of sudden increases. We are not completely sure about the cause of this phenomenon, but we suspect that at 1.6 million iteration, gradient descent helps the model enter a new landscape of parameter configurations that wasn't explored before, and it might be related to the decrease of learning rates in Adadelta.

Comparing implementation 5 and implementation 6, we can see that the increase in performance from a combination of increasing the number of heads and adding masking does not seem to be large. Although due to the long training time we are not able to explore the effects of these modifications separately, we suspect that even without padding masking, the model learned to mask out the paddings during training.

6.2 The Failure of Co-attention Layer

The addition of co-attention layer does not improve the performance, but outputs an almost identical results as the baseline. This is probably because that the attention layer of the baseline has already included interaction between question and context.
The output of baseline attention layer:

$$g_i = Bi - LSTM[c_i; a_i; c_i \cdot a_i; c_i \cdot bi]$$

Here, a_i is the C2Q attention and b_i is the Q2C attention. with the context embedding c_i, C2Q attention output a_i, and the Q2C attention output b_i.
The output of the coattention layer:

$$\begin{aligned} u_t &= Bi - LSTM([d_t; c_t^D]) \\ &= Bi - LSTM([d_t; d_t * softmax(D^T Q]) \end{aligned} \tag{1}$$

with c_t^D is attention of document in light of question and d_t is attention of question in light of document.
Since the baseline attention layer has already taken C2Q and Q2C into account, and a_i, b_i is calculated in a very similar manner as c_t^D, it is not surprising that including an additional coattention layer fails to achieve further improvement.

6.3 RNet

The self-attention layer, unlike co-attention layer, focus on the Context to Context attention. Thus, it provides additional information about attention to the QA system, thus improve the performance.

However, one significant drawback of the self attention layer is the computational cost. The computation cost of one epoch increases by 30 minutes after incorporating the self attention layer. The self-Matchin Layer computes the attention of each word in the context with the whole passage, and train a linear transformation based on that. Even after replacing Bidirectional LSTM with GRU, the computation speed is 2x slower. Unsurprisingly, the complexity of the model increases as well, which, push back the overfit time from 2M to 3.5M iterations.

Both the RNet paper and the reference code achieved a F1 of 79, and EM of 71, which is significantly better than our results. The possible reasons is that the Pointer Network does not work.After implement the Pointer Network, the gradient will converge to 0 very fast. One possible explanation is the extensive use of softmax. The gradient of the softmax function can be trivial when the value is larger than 5 or smaller than -5. If there is more time, we would fix the Pointer Network and observe the performance.

7 Conclusion

In this project, we implemented several improvement of question answering system based on SQuAD database including: 1) QANet 2) coattention 3) RNet. We built the models from scratch and evaluated against the EM and F1 score. We first implemented co-attention layer, which did not improve the model performance. We then added character-embeddings to the baseline model which improved the EM score to 60.59 and F1 score to 64.17. After that we implemented QANet incrementally and eventually saw major improvements in both EM and F1 scores (64.49 and 69.62) compared to the baseline BiDAF model and BiDAF with character-embeddings. Finally, we implemented the Self Matching layer and Pointer Network described in RNet. This is implemented on the top of Character Embedding and Baseline. We tested several modifications of the RNet architecture, e.g. different gate attention recurrent network and output layer. While which the Self Matching improved the performance, the Pointer Network caused vanishing gradients. The self match layer and character embedding improve the performance to 62.06(EM) and 65.53(F1). Among all techniques, QANet gives the best performance.

For future improvements on the QANet, we could make use of data augmentation through translating English text to French and back to English using attention-based neural machine translation models. Another possible improvement is to finish the pointer network of RNet. It can also be useful to conduct more hyper parameters tuning on the hidden size, learning rate, and etc.

References

[1] https://paperswithcode.com/sota/question-answering-on-squad11.

[2] Ali Farhadi Minjoon Seo, Aniruddha Kembhavi and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. 2016.

[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. *arXiv preprint arXiv:1804.09541*, 2018.

[4] Victor Zhong Caiming Xiong and Richard Socher. Dynamic coattention networks for question answering. 2016.

[5] R-net: Machine reading comprehension with self-matching networks.

[6] Knowledge Computation Group@HKUST. https://github.com/HKUST-KnowComp/R-Net.

[7] Jing Jiang Shuohang Wang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905, 2016b*, 2016.

[8] Meire Fortunato Oriol Vinyals and Navdeep Jaitly. Pointer networks.

[9] PyTorch. https://pytorch.org/tutorials/beginner/transformer$_tutorial.html$.

[10] andy840314. https://github.com/andy840314/QANet-pytorch.

A Appendix (optional)

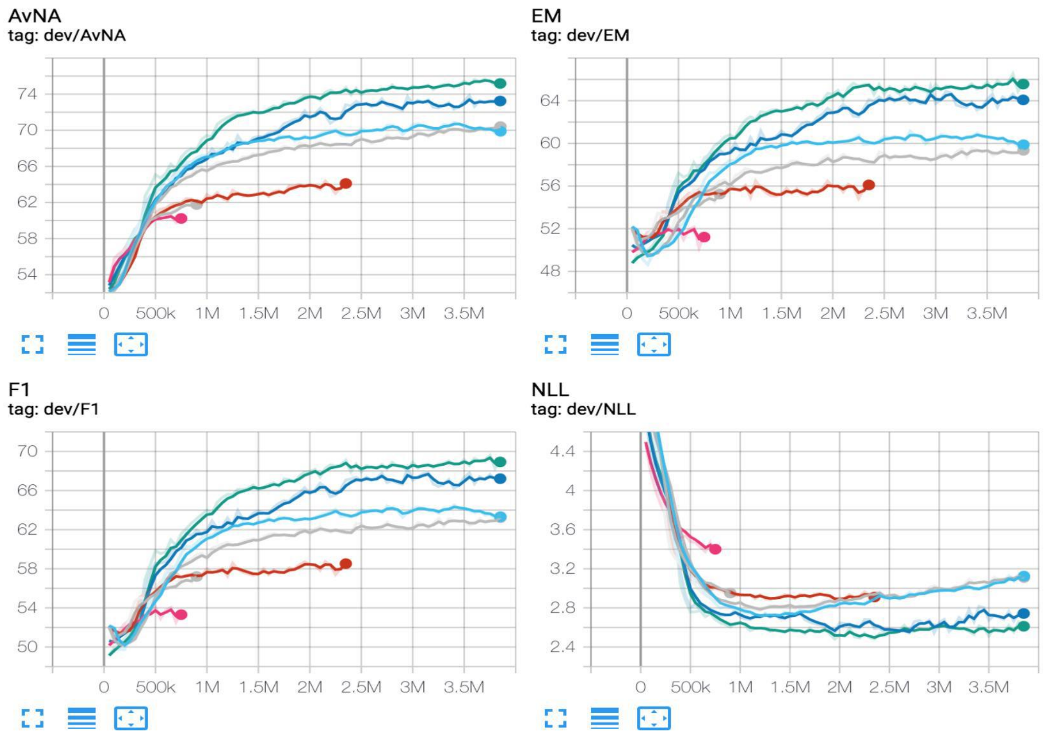

Figure 1: Character-level embedding: light blue, implementation 1: light grey right below light blue, implementation 2: short light grey, implementation 3: red, implementation 4: pink, implementation 5: dark blue, implementation 6: green

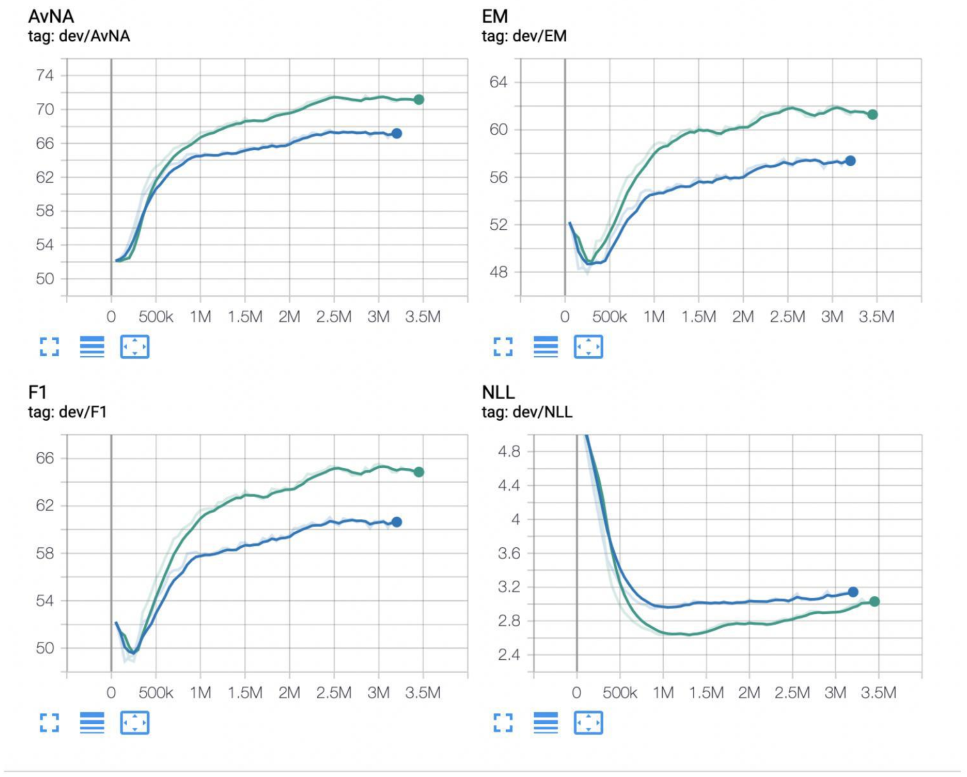

Figure 2: Visualization of Self Match + Char Embedding. The green curve is for Self Match + Char embedding, while the blue one is the baseline.