Exploring Attention Mechanisms on SQuAD 2.0

Stanford CS224N Default Project
Track: IID SQuAD

Clara Zou
Department of Statistics
Stanford University
yzou66@stanford.edu

Yichen Liu
Department of Statistics
Stanford University
yliu21@stanford.edu

Sibei Zhang
Department of Education
Stanford University
sibei@stanford.edu

Abstract

In this project, we attempt to address the question-answering task on Stanford Question Answering Dataset 2.0 (SQuAD 2.0). The goal is to build a question-answering system based on SQuAD 2.0 dataset that could correctly answer a given question based on a given context. We use BiDAF model as the baseline model, aiming to improve its performance, explore different attention techniques, and compare their performances. Our best single model that utilizes a different gate within the self-attention layer and fine-tuned hyperparameters achieves an **EM of 60.118 and an F1 of 63.866 on test set**.

1 Key Information to include

- Mentor: Fenglu Hong
- External Collaborators (if you have any): No
- Sharing project: No

2 Introduction

2.1 Motivation

For decades, Reading comprehension (RC), or the capability to process document texts and answer questions about them is a difficult task for machines, as human language understanding and real-world knowledge are needed [1]. However, the rapid growth of neural network in the past decade has brought dominant breakthroughs in text translation, speech recognition, credit card fraud detection etc. as well as many other fields that are inaccessible previously. The accelerated development of deep learning enables us to tackle with challenges in machine translation through making full use of neural network architecture and techniques. In this report, we are taking the SquAD 2.0 challenge, aiming to build a neural network that is capable of answering questions based on contexts from Wikipedia articles.

2.2 Key ideas and results

Since the invention of Transformers[2], attention mechanisms became well-implemented in many subsequent models. For our project, instead of implementing the many complicated and well-established models using self-attention, we want to investigate different attention mechanisms including coattention and self-attention by adding them to the provided BiDAF baseline model. With the final goal to improve the performace of the model compared with that of the baseline model, we attempt to do so in the following ways: adding coattention mechanism, implementing self-attention mechanism inspired by Microsoft's R-NET paper, exploring layer normalization and scaled dot product, and further fine tuning the hyperparameters. After training 9 BiDAF-based models with different designs of layers and hyperparameters, we improve the dev EM to 61.822 and the dev F1 to 65.157, and subsequently push the test EM to 60.118 and the test F1 to 63.866.

3 Related Work

Given a BiDAF baseline model, We explore character-level embeddings, self-attention layer and coattention layer along with layer-normed scaled dot product self-attention to see if any of them improve the EM and F1 scores. Among all the methods we implement, we mainly use the following four papers as references:

1. Bi-directional Attention Flow For Machine Comprehension[3], which provides the baseline model and the instruction to add character-level embeddings.

2. R-NET: Machine reading comprehension with self-matching networks[4], which is the main source of self-attention mechanism.

3. Dynamic coattention networks for question answering [5], which provides inspiration for adding coattention layer.

4. Layer Normalization [6], which, along with the lectures, adds insights on how and when to implement layer normalization and scaled dot product attention

Our work is motivated to be built upon the utilization of methods mentioned in these papers because some models, for example R-NET, achieved state of the art at some time in the past, particularly on the SQuAD 1.0 dataset and we are interested in seeing how models using similar mechanisms will perform on the new SQuAD 2.0 dataset.

4 Approach

Our explorations are all implemented based on the BiDAF baseline model[3] with four major attempted trials that we add: character-level embeddings, coattention, self-attention with different variations and hyperparameter tuning. Among these, character-level embedding, self-attention, and hyperparameter tuning achieved better results.

4.1 Character-Level Embeddings

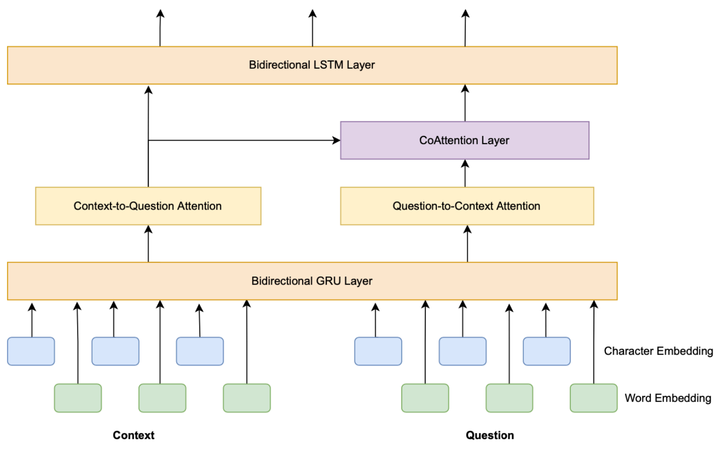
We add character-level embeddings in the embeddings layer configured in the original BiDAF paper. Given $\{\mathbf{c}_1, ..., \mathbf{c}_T\}$, and $\{\mathbf{q}_1, ..., \mathbf{q}_J, \}$ words in the context and the question, respectively, we obtain the character-level embeddings from GloVe, pass through a 1-dimensional CNN with max pooling applied on the last dimension, and concatenate with word embeddings:

$$\mathbf{c_{char}}, \mathbf{q_{char}} = \text{CNN}(\{\mathbf{c}_1, ..., \mathbf{c}_T\}, \{\mathbf{q}_1, ..., \mathbf{q}_J\})$$

$$\mathbf{c} = [\mathbf{c_{word}}, \mathbf{c_{char}}], \qquad \mathbf{q} = [\mathbf{q_{word}}, \mathbf{q_{char}}]$$

4.2 Coattention

We implement a simple version of coattention layer which does not include the sentinel vectors that are mentioned in the DCN paper[5]. The co-attention layer attends to both question and context and computes two levels of attention. The layer receives context and question hidden states.



1) First, we pass question hidden states through a linear layer with Tanh activation function:

$$\mathbf{q'}_j = \tanh(\mathbf{W}\mathbf{q}_j + \mathbf{b}) \in \mathbb{R}^l \quad \forall j \in \{1, ..., M\}$$

2)We then compute the affinity matrix L, which contains pairwise affinity score:

$$\mathbf{L}_{ij} = \mathbf{c}_i^\top \mathbf{q'}_j \in \mathbb{R}$$

3) We use softmax to normalize the matrix row-wise. The Context-to-Question coefficients are then multiplied with question hidden states to obtain attention contexts:

$$\alpha^i = \text{softmax}(\mathbf{L}_{i,:}) \in \mathbb{R}^M, \qquad \mathbf{a_i} \quad = \sum_{j=1}^{M} \alpha_j^i \mathbf{q'}_j \in \mathbb{R}^l$$

4) Similarly, the matrix is normalized column-wise to obtain coefficients. We multiply it with context hidden states to compute Question-to-Context attention output:

$$\beta^j = \text{softmax}(\mathbf{L}_{:,j}) \in \mathbb{R}^N, \qquad \mathbf{b_j} \quad = \sum_{i=1}^{N} \beta_j^i \mathbf{c}_i \in \mathbb{R}^l$$

5) The second level attention is then computed by taking sum of Question-to-Context attention weighted by Context-to-Question coefficients:

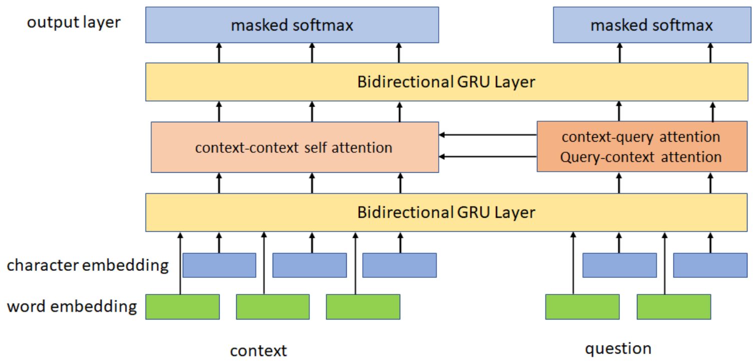
$$\mathbf{s}_i = \sum_{j=1}^{M} \alpha_j^i \mathbf{b}_j \in \mathbb{R}^l \quad \forall i \in \{1, ..., N\}$$

6) Finally, we concatenate second level attention and Contest-to-Question outputs and pass it into an single layer bidirectional LSTM:

$$\{\mathbf{u}_1, ...\mathbf{u}_N\} = \text{biLSTM}(\{[\mathbf{s}_1; \mathbf{a}_1], ...[\mathbf{s}_N; \mathbf{a}_N]\})$$

4.3 Self-Attention

Self-matching attention layer is implemented to deal with limitation in knowledge of context, which will impede us from inferring the answer. The self-attention layer attention makes each word attend to all words in the context passage to get better knowledge of the context. We use a bidirectional RNN (GRU) to build self-attention layer for computation efficiency. Two types of self-attention are explored, additive attention[4] and layer-normed scaled dot product attention[6]:



4.3.1 Additive attention

- First, we calculate the attention-pooling vector of the entire context c_t:

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

- Two types of transformation mechanism are applied:
  - Gated attention:
$$g_t = \text{sigmoid}(W_g[u_t^p, c_t]), \qquad [u_t^P, c_t]^* = g_t \odot [u_t^p, c_t]$$
  - Sigmoid & linear transformation:
$$[u_t^P, c_t]^* = \text{sigmoid}(W_g[u_t^p, c_t])$$
- Then, we calculate the self-attention layer:
$$h_t^P = BiRNN(h_{t-1}^P, [v_t^P, c_t]^*)$$

4.3.2 Layer-normed scaled dot product attention

- Normalize the hidden state:
$$x^{l'} = \frac{c^l - \mu^l}{\sigma^l + \epsilon}$$
- Obtain query, key, and value and calculate scaled dot production attention:
$$Q = W_q X, \qquad K = W_k X, \qquad V = W_v X, \qquad d_k = \text{hidden size}$$
$$c_t = \text{softmax}\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V$$
- We compute gated scaled dot product attention:
$$g_t = \text{sigmoid}(W_g[u_t^p, c_t]), \qquad [u_t^P, c_t]^* = g_t \odot [u_t^p, c_t]$$
- Then, we calculate the self-attention layer:
$$h_t^P = BiRNN(h_{t-1}^P, [v_t^P, c_t]^*)$$

5 Experiments

5.1 Data

The experiment dataset used in this project is the Stanford Question Answering Dataset (SQuAD 2.0), which contains (context, question, answer) triples. Contexts are excerpts from Wikipedia and the answer is a span of text from the context. There are 129,941 examples in training set, 6078 samples in dev set (roughly half of the official dev set) and 5915 examples in test set (remaining examples in official dev set). The training set has one answer per question, while dev and test set have three human-provided answers per question.

5.2 Evaluation method

We use two metrics to evaluate our model: Exact Match (EM) score and F1 score. EM score is a binary measure of whether the model produces answer that exactly matches the ground truth answer. A less strict metric F1 is also considered to take both precision and recall into account. When evaluating our dev set or test set, we compare the answer given by our model and three human-provided answers and take the maximum of F1 and EM score.

5.3 Experimental details

5.3.1 Baseline

We first train the baseline BiDAF model with 30 epochs, with the default setting of batch size = 64, dropout probability = 0.2, learning rate = 0.5 and hidden size = 100. The baseline model requires about 7 minutes per epoch on Azure NC6 VM.

5.3.2 Character-Level Embeddings

Then we train the BiDAF model with character-level embedding layer with 30 epochs, with hidden size = 200 after concatenation, and everything else kept the same as the baseline. The time it takes to train each epoch for BiDAF model with character-level embedding layer is about double that of the baseline.

5.3.3 Coattention Layer

We train the BiDAF model with both character embeddings and the co-attention layer, with batch size = 64, dropout probability = 0.2, learning rate = 0.5 and hidden size = 200. It takes about 12 minutes per epoch on Azure NC6 VM.

5.3.4 Self-Attention Layer

1) We train the BiDAF model with both character embeddings and self-attention layer (without gate), with batch size = 16, dropout probability = 0.2, learning rate = 0.5 and hidden size = 200. It takes about 14 minutes per epoch on Azure NC6 VM. To be able to fit the model in memory, we reduced the batch size from 64 to 16, and the training time is not affected too much by this change. It takes about 17 minutes per epoch on Azure NC6 VM.

2) We experiment with passing the self-attention layer with an additional gate[4], with the same setting as the previous trial. It takes about 17 minutes per epoch on Azure NC6 VM. The EM score on dev set increases by about 1.

3) Instead of passing self-attention layer into the gate, we experiment with using sigmoid & linear transformation on self-attention score before concatenation, which is equivalent to a gate without the hardamard product. We use the same configurations with the previous trial for better comparison. It takes about 17 minutes per epoch on Azure NC6 VM. The EM score on dev set increases by about 2 compared to raw self-attention layer.

4) In case that the input from the layers beneath has a large and uninformative variation, we adopt the layer-normed scaled dot product attention mechanism in place of the original self-attention layer, hoping to stabilize the hidden state dynamics, provided that it is indeed chaotic. We experiment with layer-normed scaled dot product attention, still with same configuration as previous trial. The training process was pre-terminated due to unpromising training curves.

5.3.5 Hyperparameter Tuning

After our explorations character-level embeddings and multiple attention mechanism, we perform hyperparameter tuning on our best-performing model: BiDAF baseline model with character-level embeddings and self-attention layer with sigmoid & linear transformation (batch size = 16, dropout probability = 0.2)

1) *Learning rate*: $\{0.5, 0.25, 0.1\}$. (Fix hidden size = 200)
2) *Hidden size*: $\{200, 300\}$. (Fix learning rate = 0.5)

5.4 Results

5.4.1 Performance on dev set

	EM	F1
BiDAF baseline (lr = 0.5)	57.335	60.803
BiDAF + Char embeddings (lr = 0.5)	58.847	62.396
BiDAF + Char embeddings + Co-attention (lr = 0.5)	52.193	52.193
BiDAF + Char embeddings + Self-attention (lr = 0.5)	57.402	60.512
BiDAF + Char embeddings + gated Self-attention (lr = 0.5)	58.427	61.472
BiDAF + Char embeddings + Self-attention with transformation (lr = 0.5)	59.435	62.756
BiDAF + Char embeddings + Self-attention with transformation (lr = 0.25)	61.822	65.157
BiDAF + Char embeddings + Self-attention with transformation (lr = 0.1)	61.838	64.993
BiDAF + Char embeddings + Self-attention with transformation (lr = 0.5, hidden size=150)	58.931	62.107

5.4.2 Performance on test set

We adopted BiDAF model with character-level embeddings and self-matching attention layer (with sigmoid & linear transformation) and learning rate = 0.25, hidden size = 200, batch size = 16 as our final model. The final model achieved performance of **EM = 60.118, F1 = 63.866 on test set**.

5.4.3 Quantitative Analysis

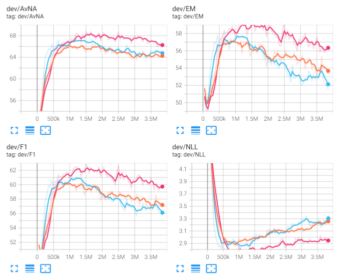


Figure 1: Gate v.s. Sigmoid & linear transformation
Pink: Sigmoid & linear transformation
Blue: Original gate

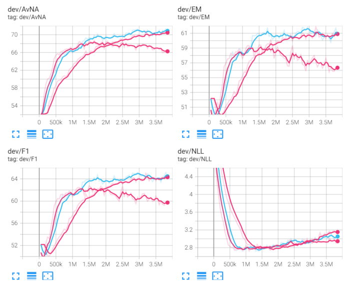


Figure 2: Learning Rate
Pink with better performance: learning rate = 0.1
Pink with weaker performance: learning rate = 0.5
Blue: learning rate = 0.25

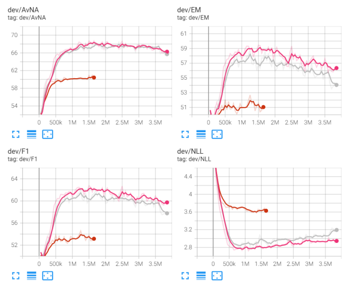


Figure 3
Pink: Gated self-attention
Red: Scaled dot product attention

1) Self-attention boosts the performance of the model by integrating knowledge from context into hidden states. To our surprise, we found that using sigmoid & linear transformation on attention layer would achieve better EM and F1 scores than the original gate mentioned in RNET paper[4]. Upon further checking the corresponding values, we discovered that hidden state values using the original gate would shrink more compared with those using sigmoid & linear transformation, which might be a pattern specific to this project.
2) Meanwhile, the fact that lower learning rate performs better than the original learning rate at 0.5 suggests that this dataset favors slower update. The original learning rate makes gradient updates excessively large, leading to divergent updates patterns that miss the minimum point.
3) The layer-normed scaled dot product attention implementation that we terminated during training due to its worse training curve than other models indicates that this mechanism is not suitable for this task. Some potential reasons include: the order of operations within the layer normalization

may not be optimal; the hidden state dynamics in this task are stable already and the adoption of this mechanism turns to be a perturbation rather than reinforcement.

6 Analysis

For a more detailed qualitative analysis, we describe some remarkable parallels between the baseline model and the final best model and discuss the typical successes and failures in each model.

6.1 Dependencies and Internal Structure within the Context

Question:	What complexity class is characterized by a computational tasks and efficient algorithms?
Context:	The complexity class P is often seen as a mathematical abstraction modeling those computational tasks that admit an efficient algorithm. This hypothesis is called the Cobham–Edmonds thesis. The complexity class NP, on the other hand, contains many problems that people would like to solve efficiently, but for which no efficient algorithm is known, such as the Boolean satisfiability problem, the Hamiltonian path problem and the vertex cover problem. Since deterministic Turing machines are special non-deterministic Turing machines, it is easily observed that each problem in P is also member of the class NP.
Answer:	P
Baseline Prediction:	N/A
Best Model Prediction:	complexity class P

In the above example, we witness that the baseline model fails to identify the answer "p" even if the first sentence of the context provides sufficient information to derive the correct answer. We believe that the presence of the word "efficiently" later in the paragraph describing the goal of people to solve the complexity class NP confuses the model. The last sentence stating that "each problem in P is also member of the class NP" may also contribute to this impact of confusion. The final best model, on the other hand, successfully predicts the correct answer due to the addition of the self-attention mechanism that enables the model to be aware of internal structures and relations between each word in the context as we believe.

6.2 N/A Prediction

Question:	Where did China border Kublai's territory?
Context:	Instability troubled the early years of Kublai Khan's reign. Ogedei's grandson Kaidu refused to submit to Kublai and threatened the western frontier of Kublai's domain. The hostile but weakened Song dynasty remained an obstacle in the south. Kublai secured the northeast border in 1259 by installing the hostage prince Wonjong as the ruler of Korea, making it a Mongol tributary state. Kublai was also threatened by domestic unrest. Li Tan, the son-in-law of a powerful official, instigated a revolt against Mongol rule in 1262. After successfully suppressing the revolt, Kublai curbed the influence of the Han Chinese advisers in his court. He feared that his dependence on Chinese officials left him vulnerable to future revolts and defections to the Song.
Answer:	N/A
Baseline Prediction:	northeast
Best Model Prediction:	N/A

Again, in this above example, the correct answer is N/A since the context does not provide enough information to answer the question even though the context is related to the question. We observe that the baseline model is confused and gives an answer distracted by the sentence "Kublai secured the northeast border in 1259," which is another example that the model gets lost by the internal structure of the context. The final best model fixes this issue and predicts "N/A" correctly with the assistance from the added self-attention layer to better understand the within-context information. This example is noteworthy since the existence of unanswerable questions is the main difference between SQuAD 2.0 that our model is trained and tested on and SQuAD 2.0 that the original baseline model and R-NET model were based on. Therefore, we discover that self-attention is a practical recipe that adapts to this new feature.

6.3 Other Languages

Question:	Who did the Han Japanese want to help the Mongols fight?
Context:	Many Han Chinese and Khitan defected to the Mongols to fight against the Jin. Two Han Chinese leaders, Shi Tianze, Liu Heima (劉黑馬, Liu Ni), nd the Khitan Xiao Zhala (蕭札剌) defected and commanded the 3 Tumens in the Mongol army. Liu Heima and Shi Tianze served Ogödei Khan. Liu Heima and Shi Tianxiang led armies against Western Xia for the Mongols. There were 4 Han Tumens and 3 Khitan Tumens, with each Tumen consisting of 10,000 troops. The three Khitan Generals Shimobeidier (石抹孛迭兒), Tabuyir (塔不已兒) and Xiaozhacizhizizhongxi (蕭札剌之子重喜) commanded the three Khitan Tumens and the four Han Generals Zhang Rou, Yan Shi,Shi Tianze, and Liu Heima commanded the four Han tumens under Ogödei Khan.
Answer:	N/A
Baseline Prediction:	Jin
Best Model Prediction:	Jin

We see that in the example above, both the baseline model and the final best model fails to predict the correct answer. We believe that this is due to the fact that most paragraphs used in training are originated from pure English contexts. The context in this example, however, has Chinese names both in pinyin and in traditional Chinese, the latter of which probably provides little information since it is outside the vocabulary. Therefore, one potential direction to improve the model is enhancing its capability of predicting the correct answer based on contexts relating to other languages and cultures.

7 Conclusion

After training 9 BiDAF-based models with different designs of layers and hyperparameters, we push the dev EM to 61.822 and the dev F1 to 65.157, and subsequently push the test EM to 60.118 and the test F1 to 63.866. In conclusion, the best of our attempted architectural changes with fine-tuned hyperparameters results in better performance than the baseline. The final best model behaves much better handling unanswerable questions and understanding the internal structure of context paragraphs.

Still, this research entails certain implications for future explorations, including more rigorous surveys on why the sigmoid & linear transformation behaves better than the original gate and the rationale behind the unsatisfactory performance of coattention and layer-normed scaled product self attention. Since the final model performs not very well on contexts relating to other languagues and cultures, it would also be worth examining with more details to improve the model on this task in potential future works.

References

[1] Konstantin Lopyrev Pranav Rajpurkar, Jian Zhang and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv*, 2016.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[3] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bi-directional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[4] Natural Language Computing Group Microsoft Research Asia. R-net: Machine reading comprehension with self-matching networks. May 2017.

[5] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

[6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.