Question Answering with Attention

Stephanie Dong
Stanford University
sxdong11@stanford.edu

Ziyi Li
Stanford University
ziyil@stanford.edu

Abstract

Machine comprehension (MC) has been a seminal task in the field of Natural Language Processing. In this paper, we explore various deep neural network components and their effectiveness when applied to a standardized MC task, via the Stanford Question Answering Dataset (SQUAD).

1 Introduction

Question answering is an important task in natural language processing. Many research work has been done in building effective reading comprehension systems for SQuAD. The SQuAD webpage (https://rajpurkar.github.io/SQuAD-explorer/) shows the performance of many systems. The top performance models use Reinforced Mnemonic Reader[1], R-net [2], and other form of attention. In January 2018, a model surpasses human performance for the first time.

In this project, we explored various deep neural network components and their effectiveness when applied to question answering. We implemented models with coattention network, and then experimented with adding self-attention layer or pointer network. We also studied the effect of tuning different hyperparamters. In the end, we analyzes our results and different types of errors.

2 Data

The SQUAD dataset consists of a public training set and public dev set. There is, in addition, a private test set on which trained models are evaluated when submitted to the SQUAD competition. Combined the datasets compose of over 100,000 questions gathered by crowdworkers contexted on a set of Wikipedia articles. The problem of predicting an answer is constructed as selecting a continuous span within the context paragraph that best answers a question.

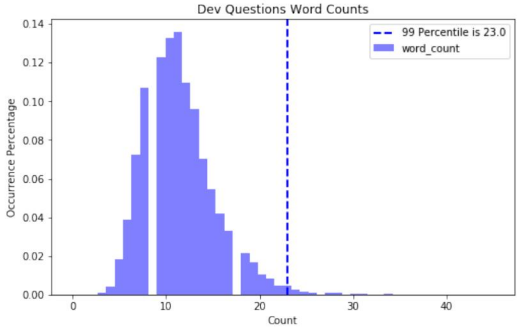
We performed numerical analysis on the train and dev question, context and answer datasets. Table 1 shows the word count of the 99^{th} percentile in each dataset.

Dataset	99 Percentile Word Count
Train Question	23
Dev Question	23
Train Context	325
Dev Context	376
Train Answers	21
Dev Answers	18

Table 1: 99 percentile word count of Train, Dev datasets.

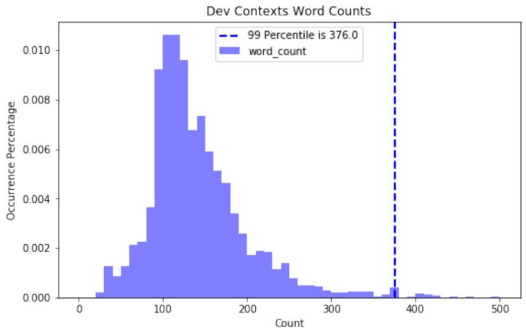
We additionally analyzed the distribution of word contexts. As shown in Figure 1, the word count distribution of context and questions follows a skewed Gaussian distribution, where as the distribution of answer length decreases exponentially from 1. Since over 55% of answers are one to two

words, most questions in the dataset can be considered ”simple.” An analysis on the distribution of



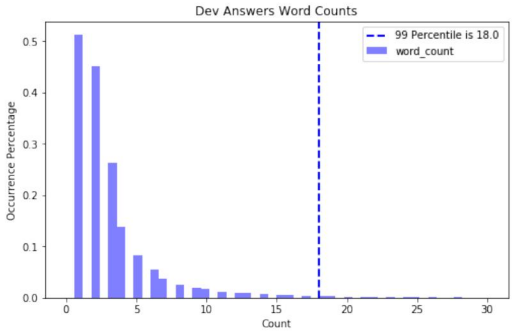
Dev Questions Word Counts

Count	Occurances	Rate
11.0	1271.0	0.127
10.0	1242.0	0.124
9.0	1148.0	0.115
12.0	1024.0	0.102
8.0	999.0	0.1
13.0	898.0	0.09
7.0	677.0	0.068
14.0	658.0	0.066
15.0	509.0	0.051
16.0	390.0	0.039
6.0	364.0	0.036
17.0	308.0	0.031
18.0	198.0	0.02
5.0	169.0	0.017
19.0	155.0	0.015



Dev Contexts Word Counts

Count	Occurances	Rate
98.0	182.0	0.093
116.0	172.0	0.088
125.0	158.0	0.081
102.0	135.0	0.069
111.0	134.0	0.068
107.0	126.0	0.064
119.0	119.0	0.061
124.0	119.0	0.061
114.0	118.0	0.06
103.0	118.0	0.06
108.0	118.0	0.06
99.0	117.0	0.06
117.0	117.0	0.06
105.0	116.0	0.059
104.0	113.0	0.058

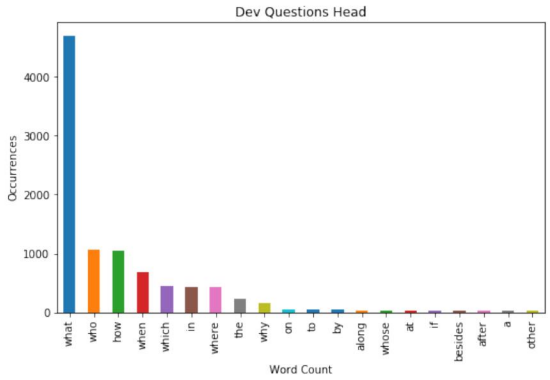


Dev Answers Word Counts

Count	Occurances	Rate
1.0	3187.0	0.312
2.0	2799.0	0.274
3.0	1634.0	0.16
4.0	859.0	0.084
5.0	515.0	0.05
6.0	339.0	0.033
7.0	228.0	0.022
8.0	158.0	0.015
9.0	119.0	0.012
10.0	106.0	0.01
11.0	68.0	0.007
12.0	64.0	0.006
13.0	54.0	0.005
14.0	42.0	0.004
15.0	30.0	0.003

Figure 1: Distribution of word count on Dev context, question and answer datasets.

the first word in in the question dataset shows us the distribution over the type of questions. Over 50% of the question start with ”What”.



Dev Questions Top Head Words

Head	Occurances	Rate
what	4704	0.5
who	1057	0.112
how	1045	0.111
when	676	0.072
which	451	0.048
in	436	0.046
where	431	0.046
the	233	0.025
why	150	0.016
on	42	0.004
to	42	0.004
by	38	0.004
along	36	0.004
whose	34	0.004
at	31	0.003

Figure 2: Distribution of first word of questions in the development set.

3 Model Components

3.1 RNN Encoder Layer

Starting from the baseline model provided in *CS 224N Default Final Project Handout*, we built the RNN encoder layer with LSTM. Let the context be represented by a sequence of d-dimensional word embeddings $[x_1, ..., x_N] \in \mathbb{R}^d$, and the question be represented by $[y_1, ..., y_N] \in \mathbb{R}^d$. Then we use bidirectional LSTM to produce a sequence of forward hidden states and a sequence of backward states.

$$\{\vec{c_1}, \overleftarrow{c_1}, ..., \vec{c_N}, \overleftarrow{c_N}\} = \text{biLSTM}(\{x_1, ..., x_N\})$$

$$\{\vec{q_1}, \overleftarrow{q_1}, ..., \vec{q_N}, \overleftarrow{q_N}\} = \text{biLSTM}(\{y_1, ..., y_N\})$$

Next, we concatenate the forward and backward hidden states to obtain context hidden states c_i and the question hidden states q_j:

$$c_i = [\vec{c_i}; \overleftarrow{c_i}] \in R^{2h}, \forall i \in \{1, ..., N\}$$

$$q_j = [\vec{q_j}; \overleftarrow{q_j}] \in R^{2h}, \forall j \in \{1, ..., M\}$$

3.2 Coattention Layer

We implemented the coattention layer from Dynamic Coattention Network [3]. It attends to the question and the context simultaneously, and then fuses both attention contexts.
First, we obtain projected question hidden states $q'_j = tanh(Wq_j + b) \in \mathbb{R}^{2h}$. Then we add sentinel vectors $c_\emptyset, q_\emptyset$ which allows the model to not attend to any particular word in the input. Next, we compute the affinity score for each pair (c_i, q'_j) of context and question hidden states:

$$L_i j = c_i^T q'_j, \text{where } L \in \mathbb{R}^{(N+1)\times(M+1)}$$

With the affinity matrix, we can compute attention outputs for both directions. For Context-to-Question (C2Q) Attention, we calculate the outputs a_i as follows:

$$\alpha^i = softmax(L_{i,:}) \in \mathbb{R}^{M+1}, a_i = \sum_{j=1}^{M+1} \alpha_j^i q'_j \in \mathbb{R}^{2h}$$

Similarly, for Question-to-Context (Q2C) Attention, we calculate the outputs b_j as follows:

$$\beta^j = softmax(L_{:,j}) \in \mathbb{R}^{N+1}, b_j = \sum_{i=1}^{N+1} \beta_i^j c_i \in \mathbb{R}^{2h}$$

Then we can calculate the second-level attention outputs s_i as the weighted sums of the Q2C and C2Q outputs.

$$s_i = \sum_{j=1}^{M+1} \alpha_j^i b'_j \in \mathbb{R}^{2h}$$

The final step is the fusion of temporal information to the second-level attention outputs via a bidirectional LSTM. The overall output of the Coattention Layer is:

$$\{\mathbf{u}_1, ..., \mathbf{u}_N\} = \text{biLSTM}(\{[s_1; a_1], ..., [s_N; a_N]\})$$

3.3 Self Attention Layer

We implemented the a modified version of Self Matching Attention layer from R-Net [2] using dot-product attention instead of additive attention. It attends the context to itself.

First we obtain context representations from the previous network architecture component $\{\mathbf{v}_1, ..., \mathbf{v}_N\} \in \mathbb{R}^\ell$, where each $\mathbf{v}_i$ represents the context location $i \in \{1, ..., N\}$.

$$\mathbf{e}_j^i = \text{ReLU}(\mathbf{W}_1\mathbf{v}_j)^T\text{ReLU}(\mathbf{W}_2\mathbf{v}_i) \in \mathbb{R}$$
$$\alpha^i = \text{softmax}(\mathbf{e}^i) \in \mathbb{R}^N$$
$$\mathbf{a}_i = \sum_{j=1}^{N} \alpha_j^i \mathbf{v}_j \in \mathbb{R}^\ell$$

Here $\mathbf{W}_1$ and $\mathbf{W}_2$ are trainable weights. The overall output of the Self Attention Layer is:

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

3.4 Softmax Output Layer

We apply a fully connected layer with ReLU non-linearity to each output of the previous network architecture layer and apply a ReLU non-linearity.

$$\mathbf{b}'_i = \text{ReLU}(\mathbf{W}_{FC}\mathbf{u}_i + \mathbf{v}_{FC}) \quad \in \mathbb{R}^h \qquad \forall i \in \{1, ..., N\}$$

Here $\mathbf{W}_{FC}, \mathbf{w}_{\text{start}}, \mathbf{v}_i$ and $\mathbf{u}_{\text{start}}$ are trainable weights. We then apply a softmax layer to produce probability distribution over start indicies.

$$\text{logits}_i^{\text{start}} = \mathbf{w}_{\text{start}}^T\mathbf{b}'_i + \mathbf{u}_{\text{start}} \quad \in \mathbb{R} \qquad \forall i \in \{1, ..., N\}$$
$$p^{\text{start}} = \text{softmax}(\text{logits}^{\text{start}}) \quad \in \mathbb{R} \qquad \forall i \in \{1, ..., N\}$$

We compute a probability distribution p^{end} the same way, with $\mathbf{w}_{\text{end}}$ and $\mathbf{u}_{\text{end}}$

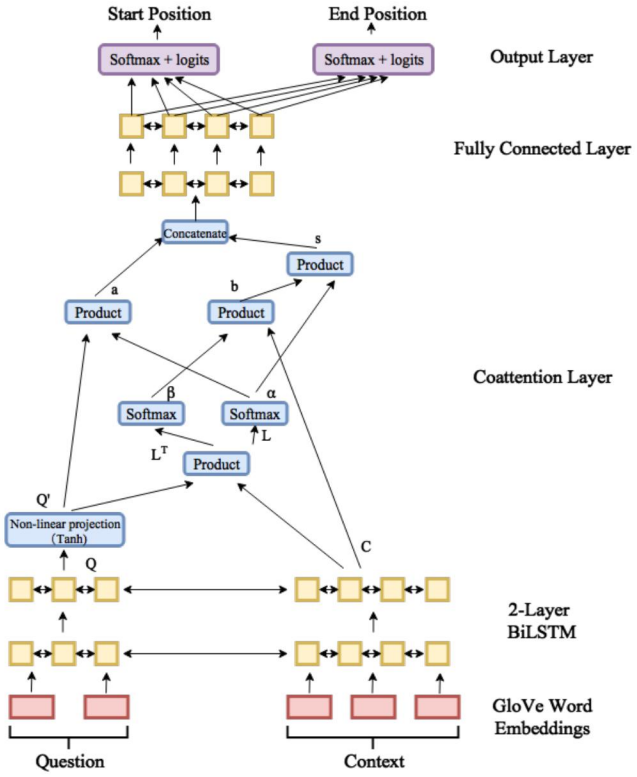


Figure 3: Architecture of Our Model with Coattention Network

3.5 Pointer Network Output Layer

We implemented the Pointer Network layer from R-Net [2], which depends on the predicted start index distribution to predict the end index distribution.

We first compute the hidden state at time zero via attention on the question hidden representation.

$$
\begin{aligned}
s_j &= v_q^T \tanh(\mathbf{W}_q q_j) \\
a_i &= \text{softmax}(s) \\
h_0 &= \sum_{i=1}^{M} a_i q_j
\end{aligned}
$$

We then compute the start distribution via additive attention on the context representation of the last layer, $\{v_1, ..., v_N\} \in \mathbb{R}^\ell$, where each v_i represents the context location $i \in \{1, ..., N\}$ and h_0.

$$
\begin{aligned}
s'_j &= v_s^T \tanh(\mathbf{W}_c v_j + \mathbf{W}_h h_0) \\
p^{\text{start}} &= \text{softmax}(s')
\end{aligned}
$$

We then compute the next hidden state via p^{start} and context representations by runing a single GRU time step.

$$
\begin{aligned}
c_1 &= \sum_{i=1}^{M} p_i^{\text{start}} v_j \\
h_1 &= \text{GRU}(h_0, c_1)
\end{aligned}
$$

From h_1, we then use additive attention on the context representation to compute the end index distribution.

$$
\begin{aligned}
s''_j &= v_s^T \tanh(\mathbf{W}_c v_j + \mathbf{W}_h h_1) \\
p^{\text{end}} &= \text{softmax}(s'')
\end{aligned}
$$

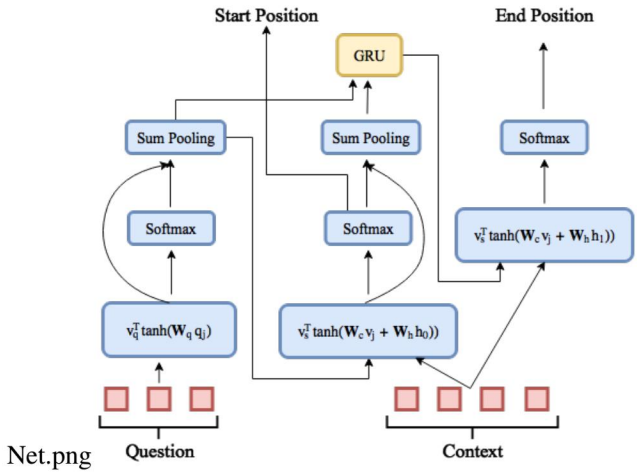


Figure 4: Pointer Network

4 Experiments

4.1 Performance Comparison of Network Architectures

Table 2 shows the performance of the models that we experimented.

Among these models, the default final output layer is the Softmax Output Layer, except for the 2-layer LSTM + Coattention + PointerNet model, which uses the Pointer Network as the final output layer.

Model	F1 Score	EM
2-layer LSTM + Coattention + PointerNet	73.057	62.498
2-layer LSTM + Coattention + Self-Attention	72.664	62.233
1-layer GRU + Coattention + Self-Attention	69.578	58.821
2-layer LSTM + Coattention	**73.488**	**62.857**
1-layer GRU + Coattention	68.569	56.547
2-layer LSTM + Basic Attention	48.279	39.612
Baseline (GRU)	43.341	34.361

Table 2: Performance Comparison on Dev Set

4.2 Test Time Optimization

We evaluated 3 different techniques for choosing the start index and end index from their respective predicted distributions. Our baseline technique was independent argmax on start distribution and end distribution.

Informed by our analysis on 99 percentile word count, we applied a dependent argmax that chose the end position based on the chosen start index. We first take the argmax over the predicted start distribution as the start index. Given the start index, we set all end index probability of indicies before the start index and all indices more than 21 words after the start index to zero, since 21 is the 99 percentile word count for the dev answers dataset. This leads to an 3 point increase in F1/EM scores over the baseline technique for our model architectures.

The third technique we explored was a K-factor beam search on the start and end distributions, returning the start index and end index with the highest probability sum. In this case, we first compute the top K argmax indices for start and end index independently, then we iterative over the K^2 possible start-end index pairs and return the pair with the highest probability sum, that satisfies the relational constraint between start and end index of the technique in the previous paragraph. We found this K-factor beam search to run much slower compared to the simple relational constraint above and only offered within a +/- 0.5 change in performance.

Based on this evaluation, we chose the second technique of simple relational constraint for our final model, for it's speed of execution and performance improvement.

4.3 Hyperparameter Tuning

4.3.1 Optimizer

We experimented with Stochastic Gradient Descent optimizer with different learning rates, e.g. 0.1, 0.5, 0.8. We found that SGD optimizer does not improve the performance. The best performance we obtained with SGD is F1/EM equals 61.555/50.927 with learning rate 0.5.

We also found that Adam optimizer usually converges faster and requires less hyperparameter tuning. The performance of SGD optimizer largely depends on the learning rate. The reason that SGD optimizer does not perform better might be that the learning rate we picked is not optimal.

4.3.2 Embedding Size

We experimented with embedding size 200, and got out-of-memory error with batch size 100. Then we used embedding size 200 with batch size 80. We obtained 60.391/49.016 as F1/EM score. Thus, increasing embedding size didn't improve the performance of our model.

4.3.3 Dropout Rate

The default dropout rate is 0.15. We see that there are overfitting in our model, so we decided to experiment with a larger dropout rate. The performance does not improve with dropout rate 0.2: F1/EM=61.3/50.237. The reason might be that the 0.2 is too high and we are adding too much regularization. We think that other regularization techniques may be helpful, such as adding l_2 norm constraints to weight vectors.

4.3.4 Context and Question Truncation Length

Informed by the analysis on word counts of context and answers, we trained a 2 layer LSTM model with CoAttention limiting the max question length and the max context length to their respective 99 percentile word counts. This resulting model performed on par with the single GRU layer CoAttention model. The same network architecture with a max context length of 600 and a max question length of 30 performed 5% better.

We observe at 99 percentile, the truncation of context and question is too aggressive to maintain performance.

4.4 Error Analysis

4.4.1 Correct Start Position but Wrong End Position

Example:

- **CONTEXT:** To remedy the causes of the fire, changes were made in the block ii spacecraft and operational procedures, the most important of which were use of a _nitrogen/oxygen_ mixture instead of pure oxygen before and during launch, and removal of flammable cabin and space suit materials. The block ii design already called for replacement of the block i _plug-type_ hatch cover with a quick-release, outward opening door.
- **QUESTION:** What type of materials inside the cabin were removed to help prevent more fire hazards in the future?
- **TRUE ANSWER:** flammable cabin and space suit materials
- **PREDICTED ANSWER:** flammable cabin and space suit materials. The block ii design

We see several examples where the predicted answer has correct start position but it's longer than the true answer. So the model fails to identify the correct end position. In the above example, it also fails to identify the end of a sentence. We think that adding interaction between start position and end position could help with this problem. For example, we can condition end position on the start position as mentioned in the handout.

4.4.2 Wrong Position of Attention

Example:

- **CONTEXT:** In early 2012, nfl commissioner roger goodell stated that the league planned to make the 50th super bowl "spectacular" and that it would be "an important game for us as a league".
- **QUESTION:** What one word did the nfl commissioner use to describe what super bowl 50 was intended to be?
- **TRUE ANSWER:** spectacular
- **PREDICTED ANSWER:** us as a league

Our model did choose an answer from the words of the NFL commissioner, but it failed to understand the meaning of "one word" in the question. With one coattention layer, it didn't capture the complex interaction between question and context. So adding more layers to recursively compute Context-to-Question and Question-to-Context attention might be helpful in this case.

5 Conclusion

We explored various neural architectures and optimization techniques on the task of end-to-end machine reading comprehension task presented by Stanford Question Answering Dataset.

From our network architecture evaluations, we can conclude that a 2-layer LSTM offers a definitive increase of F1 and EM scores by approximately 5. Likewise, Coattention offers a definitive performance improvement over basic attention. Stacking various attention components will not necessarily lead to better performance. Without a large effort in hyperparameter optimization, multiple attention components may lead to a loss of performance, as show in our results.

Truncation of max question length and max context length may be an effective technique to increase the speed of traning. However, it is easy to over truncate and lead to an overall decreased in model performance.

Upon evaluation of predicted samples for which our best model produced an incorrect answer, the majority of errors where caused by the model predicting an answer that is too detailed or not detailed enough compared to the ground truth answer. These are cases where the F1 score of the incorrect sample is non-zero, and the predicted answer is relevant enough that a human reviewer may consider the predicted answer as correct. For this class of prediction error, it would be helpful if the training dataset can be augmented with multiple alternative ground truth answers per question-context par, so the model can learn to balance between multiple forms of the "correct" choice to learn.

Another less common but much more important class of prediction error is the case where the F1 score is zero, and the model gets the answer completely wrong. Upon inspection, it appears that the majority of zero F1 score prediction have a complex question, where the model predicted some answer more related to the supporting clause of the question instead of the main subject of the question. We suspect these errors are due to the model not being able to identify the main subject of complex question. We propose that augmenting the question word embeddings with dependency parse tree of the question will help this class of errors. However, we were not able to incorporate question dependency as feature in our model architectures in a way that will train. As future work, we hope to explore different techniques of incorporating question dependency, part of speech tag, and other features to our existing model architectures and believe these additional features will lead to an increase in model performance.

Acknowledgments

We would like to thank all CS224N teaching staff for a great class experience.

References

[1] M. Hu, Y. Peng, and X. Qiu. Reinforced Mnemonic Reader for Machine Comprehension. *ArXiv e-prints*, May 2017.

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

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