Question Answering System on the Stanford Question Answering Dataset (SQuAD)

Richard Akira Heru*
Stanford University
Stanford, CA 94305
rheru@stanford.edu

Abstract

Answering a question given background information in the form of a context paragraph turns out to be a challenging task, even for humans. This project aims to reimplement various techniques to build a question answering system that works on the Stanford Question Answering Dataset (SQuAD). The best test performance achieved in this paper has an F1 score of 58.2% and an EM score of 46.4%. The model that achieves this performance comprises an RNN encoder layer, a bidirectional attention flow layer and an output layer. The start and end locations of the answer are chosen such that the end location of the answer appears at or after the start location.

1 Introduction

Reading comprehension is a difficult task, even for humans. It is even more so for machine. Unlike answering a question based on information that is short (one sentence long for instance), answering question based on a paragraph long text is more difficult as it requires the system to locate various relevant information and this information can even interact in a more complicated manner. Current advances in Deep Learning applied to Natural Language Processing have allowed machines to achieve better-than-human performance in a restricted set of question answering tasks, specifically the literal question type which appears on the Stanford Question Answering Dataset (SQuAD) [1]. The answer to the question can be found in the context paragraph for this type of question.

This project aims to reimplement various techniques to achieve this question answering task on the SQuAD. More formally, given a sequence of context words of length C, $\{x_i\}_{i=1}^{C}$, and a sequence of question words of length Q, $\{y_i\}_{i=1}^{Q}$, we want to find a function f that takes these two sequences as inputs and outputs a contiguous sequence of words which is a subsequence of the context sequence as an answer to the question given the context, i.e. $f(\{x_i\}_{i=1}^{C}, \{y_i\}_{i=1}^{Q}) = \{x_i\}_{i=\ell_{\text{start}}}^{\ell_{\text{end}}}$ where ℓ_{start} and ℓ_{end} are the start and end locations of the answer respectively with $1 \leq \ell_{\text{start}} \leq \ell_{\text{end}} \leq C$.

In this project, I experimented with various models including the baseline model, a model with bidirectional attention flow layer, a model with a smarter span selection incorporated, and a model which includes a self attention layer.

2 Related Work

As can be seen in the SQuAD explorer (https://rajpurkar.github.io/SQuAD-explorer/), there have been over 100 submissions to the official SQuAD leaderboard with varying performances. Most of these models use some form of attention layers. Some of

*Codalab username: richardakira

[1] https://rajpurkar.github.io/SQuAD-explorer/

them use bidirectional attention flow layer and self-attention layer specifically [6] [4]. The model with highest Exact Match (EM) score has an EM score of 82.8% and the model with highest F1 score has an F1 score of 89.3% (on March 20, 2018).

3 Approach

3.1 Baseline Model

The provided baseline model [1] comprises three components. The description below follows [1] closely. The first component is an RNN encoder layer. It encodes both the context and the question GloVe embeddings into hidden states. It is followed by an attention layer which combines the context and question representations. The final component is an output layer which applies a fully connected layer and then two separate softmax layers − one to get the start location and one to get the end location of the answer span.

Each SQuAD example has a context, a question, and an answer. The context is represented by a sequence of d-dimensional GloVe embeddings $\{x_i\}_{i=1}^{C}$, the question by a sequence of d-dimensional GloVe embeddings $\{y_i\}_{i=1}^{Q}$, and the answer by a pair of two gold indexes $i_{\text{start}}, i_{\text{end}} \in \{1, \ldots, C\}$ denoting the correct start and end indexes of the answer span. The GloVe embeddings are pre-trained and fixed during training. These embeddings are used as inputs for a 1-layer bidirectional GRU. The bidirectional GRU outputs a sequence of forward and backward hidden states $\vec{c_i}, \overleftarrow{c_i}, \vec{q_i}, \overleftarrow{q_i}$ for each context tokens and question tokens as shown in the formulae

$$\{\vec{c_1}, \overleftarrow{c_1}, \ldots, \vec{c_C}, \overleftarrow{c_C}\} = \text{biGRU}(\{x_i\}_{i=1}^{C})$$
$$\{\vec{q_1}, \overleftarrow{q_1}, \ldots \vec{q_Q}, \overleftarrow{q_Q}\} = \text{biGRU}(\{y_i\}_{i=1}^{Q}).$$

We then concatenate the forward and backward hidden states to get the context hidden states c_i and question hidden states q_j,

$$c_i = [\vec{c_i}; \overleftarrow{c_i}] \text{ for } i \in \{1, \ldots, C\}$$
$$q_j = [\vec{q_j}; \overleftarrow{q_j}] \text{ for } j \in \{1, \ldots, Q\}.$$

For each context hidden state c_i, we take its dot product with all question hidden states q_j and obtain the attention distribution α^i which is then used to produce the attention output a_i which is the weighted sum of the question hidden states using the following formulae

$$e^i = [c_i \cdot q_1, \ldots c_i \cdot q_Q]$$
$$\alpha^i = \text{softmax}(e^i)$$
$$a_i = \sum_{j=1}^{Q} \alpha_j^i q_j.$$

We then concatenate these attention outputs to the context hidden states to get the blended representations

$$b_i = [c_i; a_i].$$

These blended representations are then passed through a fully connected layer with ReLU non-linearity to obtain

$$b_i' = \text{ReLU}(W_{FC} b_i + v_{FC})$$

with W_{FC} and v_{FC} being the weight matrix and bias vector of the fully connected layer respectively. This is used to get scores for each context location i for start and end locations given by

$$\text{score}_i^{\text{start}} = w_{\text{start}} \cdot b_i' + u_{\text{start}}$$
$$\text{score}_i^{\text{end}} = w_{\text{end}} \cdot b_i' + u_{\text{end}}$$

where $w_{\text{start}}, w_{\text{end}}, u_{\text{start}}, u_{\text{end}}$ are the weight vectors and bias terms. These scores are then used to compute the softmax probability distributions p^{start} and p^{end} over the context locations

$$p^{\text{start}} = \text{softmax}(\text{score}^{\text{start}})$$
$$p^{\text{end}} = \text{softmax}(\text{score}^{\text{end}}).$$

These probability distributions are useful for computing loss and obtaining predictions for start and end locations. The loss is given by

$$\text{loss} = -\ln(p^{\text{start}}(i_{\text{start}})) - \ln(p^{\text{end}}(i_{\text{end}})).$$

The baseline model simply takes the indexes ℓ_{start} and ℓ_{end} such that $p^{\text{start}}(\ell_{\text{start}})$ and $p^{\text{end}}(\ell_{\text{end}})$ are maximised[2].

3.2 Bidirectional Attention Flow

The first modification to the baseline model I implemented changes the attention layer such that the attention can flow in both directions − from the context to the question and from the question to the context. This Bidirectional Attention Flow (BiDAF) originally appears in [6]. In the attention layer, we first compute the similarity matrix

$$S_{ij} = w_{\text{sim}} \cdot [c_i; q_j; c_i \circ q_j]$$

which is the dot product between the similarity weight vector w_{sim} and the concatenation of context hidden state, question hidden state and the element-wise product of context hidden state and question hidden state.

The Context-to-Question Attention is similar to our baseline model. The attention distributions α^i are obtained by taking row-wise softmax of the similarity matrix S and these are used to compute the weighted sum of the question hidden states to get the Context-to-Question attention outputs a_i using the formulae

$$\alpha^i = \text{softmax}(S_{i,:})$$
$$a_i = \sum_{j=1}^{Q} \alpha^i_j q_j.$$

In BiDAF, we also compute Question-to-Context Attention. This is done by taking the maximum element for each row and applying softmax to them to get the weight for each context hidden state. The Question-to-Context attention outputs c' are the weighted sums of the context hidden states. In equations,

$$m_i = \max_j S_{ij}$$
$$\beta = \text{softmax}(m)$$
$$c' = \sum_{i=1}^{C} \beta_i c_i.$$

We then combine all these results by concatenating them to get the blendeds representation

$$b_i = [c_i; a_i; c_i \circ a_i; c_i \circ c']$$

which are then fed into the output layer as in the baseline model.

3.3 Smarter Span Selection

Another modification pertains to the selection of start and end locations of the answer span. Instead of taking the start location with highest probability score and end location with highest probability score separately, the start and end locations are chosen such that the end location of the answer appears at or after the start location. This resolves the problem of having a start location after the end location which is certainly invalid. Similar to [3], ℓ_{start} and ℓ_{end} are chosen such that they satisfy $\ell_{\text{start}} \leq \ell_{\text{end}} \leq \ell_{\text{start}} + k$ for some positive integer k and $p^{\text{start}}(\ell_{\text{start}})p^{\text{end}}(\ell_{\text{end}})$ is maximised.

[2]It is possible that $\ell_{\text{start}} > \ell_{\text{end}}$ in which an empty string is the predicted answer.

3.4 Self Attention

In a self attention layer which also appears in [4] as Self-Matching Attention, given a sequence of representations $v_1, v_2, \ldots v_C$ corresponding to the context words, each representation attends to all the representations in $\{v_1, v_2, \ldots v_C\}$. To compute the attention output a_i, we use the formulae

$$
\begin{aligned}
e_j^i &= v \cdot \tanh(W_1 v_j + W_2 v_i) \\
\alpha^i &= \text{softmax}(e^i) \\
a_i &= \sum_{j=1}^{C} \alpha_j^i v_j
\end{aligned}
$$

where W_1, W_2 are weight matrices and v is a weight vector. The concatenation of the representation and the attention output is then fed into a bidirectional GRU to get a new set of hidden states

$$
\{h_1, h_2, \ldots, h_C\} = \text{biGRU}(\{[v_i; a_i]\}_{i=1}^{C}).
$$

In one of the models I experimented with, the self-attention layer appears in between the RNN encoder layer and the BiDAF attention layer. It takes in the context hidden states from the RNN encoder layer as the representations. The hidden states produced from the self-attention layer are then fed into the BiDAF layer to replace the context hidden states. This is different from the usage in R-Net [4] which performs Passage Self-Matching on top of Question-Passage Matching (Context-to-Question Attention in baseline model presented here) which is also suggested in the CS 224N Default Final Project Handout [1].

Figure 1 shows the illustration of the architecture containing an RNN encoder layer, a self-attention layer, a BiDAF layer and an output layer.

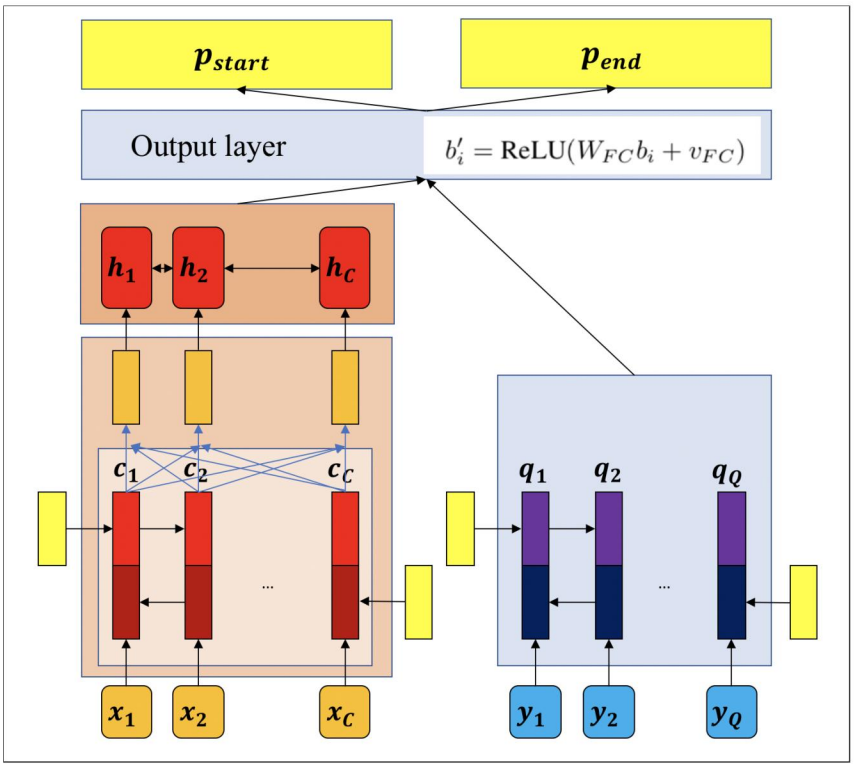

Figure 1: Illustration of the neural network architecture

4 Experiments

4.1 The Dataset

The Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, which is made of questions posed by crowdworkers based on some Wikipedia articles [5]. The answer to every question can be found in the context passage. Given the sufficiently huge size of SQuAD with over 100,000 question-answer pairs on more than 500 articles, SQuAD is suitable for deep learning models which typically require a large enough dataset to be able to perform satisfactorily.

The distributions of the context lengths, question lengths and answer lengths of all 86,318 data points in the training set are shown in Figure 2.

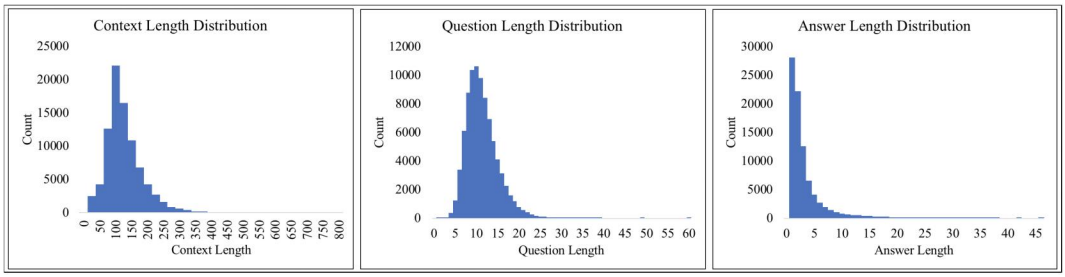

Figure 2: Distributions of context, question, and answer lengths

As we can see from the graphs, most of the contexts have lengths shorter than 400 words, most questions are shorter than 30 words and most answers are shorter than 15 words. This can motivate certain experimental design choices.

4.2 Configurations

For most of the trainings, I used the 100-dimensional GloVe embeddings. Experiments with higher dimensional embeddings (200 and 300) were attempted but they did not result in significant boost in performance. Running the model with higher-dimensional embeddings are more computationally expensive both time-wise and space-wise as it will require more parameters to train. Given the limited memory size we had, 100-dimensional GloVe was the most feasible and appropriate for our task.

My models were trained using Adam optimizer with learning rate 0.001. The training times vary, usually until the models reach their plateaus in evaluation metric (EM and F1 scores to be described in Section 4.3) on the dev set.

Regularisation was done using dropout with dropout probability of 0.15 in some cases and 0.2 in the other cases. There is no significant difference in performance resulting from the two dropout probabilities. Hidden state size was kept at 200 for most cases except for one case when it was increased to 300 resulting in negligible increase in performance. With the correct early stopping times, overfitting can be avoided. In fact, overfitting were not observed in all of my experiments as the scores on dev set did not fall, suggesting that the models generalized pretty well from train set to dev set.

The maximum question length was kept at 30 throughout all experiments and the maximum context length was initially set at 600 before it was decreased to 400 as shown in Figure 2 given the distribution of most context lengths to allow faster computation and more efficient memory usage.

Finally, the value of k chosen for the smarter span selection was set at 15 provided most answers are shorter than 15 words as can be seen in Figure 2. This means, we only consider the pairs $(\ell_{\text{start}}, \ell_{\text{end}})$ which are at most 15 words apart with $\ell_{\text{start}} \leq \ell_{\text{end}}$.

4.3 Evaluation Metric

The performance of our models are evaluated using the standard Exact Match (EM) score and F1 score.

The Exact Match (EM) score is the percentage of system outputs that match exactly with the ground truth answers. For example, if the ground truth answer is "Duka Tesla", a system output "Duka Tesla" will be considered correct and and will contribute positively to the Exact Match score whereas a system output "Tesla" will be considered strictly incorrect.

The F1 score, on the other hand, is less strict and may give some points for a system output "Tesla" which does not match the ground truth "Duka Tesla" completely. F1 score is the harmonic mean between precision and recall. Here, precision p is the fraction of predicted answer spans that line up exactly with the spans of ground truth answers. Recall r is the fraction of words in the ground truth answers that appear at exactly the same location in the predictions. The F1 score is the harmonic mean of the two, i.e. F1 $= \frac{2pr}{p+r}$.

4.4 Results and Analysis

After about 15000 iterations of training, the baseline model achieved an EM score of 29.3% and an F1 score of 40.3% on the dev set. The first improvement I made to the baseline model was changing the attention layer with BiDAF attention layer. This first improved model increased the EM score by more than 5 points to 34.9% and the F1 score by around 7 points to 47.1%. This observation is as expected as BiDAF allows the attention to flow both ways from question to answer and from answer to question unlike the baseline model which only allows the attention to flow in one direction. Replacing the GRU cells in the RNN encoder layer to LSTM cells result in worse performance with 33.4% EM score and 45.4% F1 score in my experiment. As such, subsequent experiments were all completed with GRU cells.

As can be seen in Figure 3, the baseline model (or any other models that do not change the way start and end locations are selected) immediately fails when the highest probability for end location is at a location appearing before the location with highest probability for start location.

```
CONTEXT: (green text is true answer, magenta background is predicted start, red background is predicted end, _underscores_ are u
nknown tokens). Length: 119
tesla was born on 10 july [ _o.s_ . 28 june ] 1856 into a serb family in the village of _smiljan_ , austrian empire ( modern-day
 croatia ) . his father , milutin tesla , was a serbian orthodox priest . tesla 's mother , _duka_ tesla ( née mandić ) , whose
father was also an orthodox priest , :10 had a talent for making home craft tools , mechanical appliances , and the ability to m
emorize serbian epic poems . _duka_ had never received a formal education . nikola credited his eidetic memory and creative abil
ities to his mother 's genetics and influence . tesla 's progenitors were from western serbia , near montenegro . :12
            QUESTION: what was tesla 's mother 's name ?
         TRUE ANSWER: duka tesla
    PREDICTED ANSWER:
     F1 SCORE ANSWER: 0.000
            EM SCORE: False
```

Figure 3: Example of start location appearing after end location

This motivates us to implement a smarter span selection. After the implementation of smarter span selection, the model performed better and answered the same question given the same context as in Figure 3 correctly as shown in Figure 4.

```
CONTEXT: (green text is true answer, magenta background is predicted start, red background is predicted end, _underscores_ are u
nknown tokens). Length: 119
tesla was born on 10 july [ _o.s_ . 28 june ] 1856 into a serb family in the village of _smiljan_ , austrian empire ( modern-day
 croatia ) . his father , milutin tesla , was a serbian orthodox priest . tesla 's mother , _duka_ tesla ( née mandić ) , whose
father was also an orthodox priest , :10 had a talent for making home craft tools , mechanical appliances , and the ability to m
emorize serbian epic poems . _duka_ had never received a formal education . nikola credited his eidetic memory and creative abil
ities to his mother 's genetics and influence . tesla 's progenitors were from western serbia , near montenegro . :12
            QUESTION: what was tesla 's mother 's name ?
         TRUE ANSWER: duka tesla
    PREDICTED ANSWER: duka tesla
     F1 SCORE ANSWER: 1.000
            EM SCORE: True
```

Figure 4: Example of correct prediction after smarter span selection

In terms of overall performance, the BiDAF model that was trained previously managed to get around a 4-point jump in the EM score and a 6-point jump in the F1 score on the dev set as can be seen in the jump from bidaf to bidaf_pick in Figure 5.

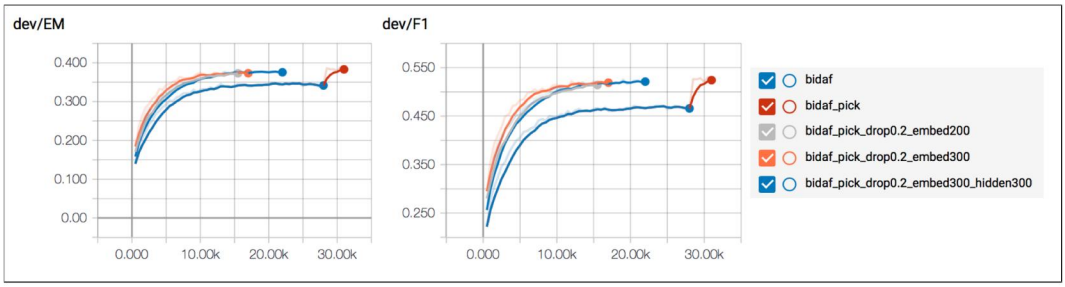

Figure 5: Performance graphs for some models

From Figure 5, we can also see that changing the dropout probability to 0.2, increasing embedding sizes to 200 or 300 and increasing hidden state dimension to 300 did not improve the performance for our BiDAF with smarter span selection. It seemed like our BiDAF model had hit its performance bottleneck and would require other improvements to increase its performance.

The next improvement pertained to the self-attention layer described earlier in Section 3.4. Unfortunately, my different implementation is not very successful as it not only performs worse in terms of the EM scores and F1 scores but also runs slower due to the additional biRNN layer in the self-attention layer. Nonetheless, it still performed slightly better than BiDAF with LSTM cell on RNN encoder layer. This gives us an insight that increasing the complexity of the model does not always result in a better overall performance. Rather, the connections within and between various layers may be more important for the working and thus the performance of the model.

A summary of performances of various trials is shown in Table 1.

Table 1: Performances of various models on dev set

Model	EM score	F1 score
Baseline model	29.3	40.3
BiDAF	34.9	47.1
BiDAF with LSTM instead of GRU	33.4	45.4
BiDAF with smarter span selection	**38.8**	**53.0**
Self-Attention + BiDAF	33.9	46.1

4.5 Other Failed Attempts

Various other attempts I have tried that gave worse-than-baseline performances include linked biRNN for context and question in the RNN encoder layer both with LSTM cells as attempted in [2] and GRU cells which gave around $8 - 9\%$ EM score after a relatively small number of iterations (around $1000 - 3000$). I stopped these models early as they did not show promising trends. This can be due to bugs in my code for these implementations or various other unknown reasons.

5 Conclusion

Overall, among all the models I have experimented with, the model with an RNN encoder layer equipped with GRU cells, bidirectional attention flow (BiDAF) attention layer, baseline output layer, and smarter span selection performed best achieving an F1 score of 58.2% and an EM score of 46.4% on the official test set.

There are still a lot of rooms for improvement. First, I could have reimplemented self-attention just as suggested in [1] similar to [4]. We can also improve other parts of the architecture other than the attention layer which had been the main focus for my project so far. For example, we can add additional features (using the Part of Speech Tagger or Name Entity Recognizer for example) to the feature vector of each word in the context and question. We can also incorporate character-level encodings and use Convolutional Neural Network to get these additional 'features.' We can also change the output layer such that it has more hidden layers.

Acknowledgments

Thank you Richard Socher for teaching this amazing class and all the TAs, especially Abi See who helped me resolve a Codalab submission issue. Thank you Microsoft for its Azure sponsorship which enabled me to train my model using GPU on NV6 VM.

References

[1] CS 224n Default Final Project: Question Answering.

[2] Budianto. Reading Comprehension on the SQuAD Dataset.

[3] D. Chen, A. Fisch, J. Weston, and A. Bordes. Reading Wikipedia to Answer Open-Domain Questions. *arXiv:1704.00051 [cs]*, Mar. 2017. arXiv: 1704.00051.

[4] Natural Language Computing Group, Microsoft Research Asia. R-NET: Machine Reading Comprehension with Self-Matching Networks.

[5] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *arXiv:1606.05250 [cs]*, June 2016. arXiv: 1606.05250.

[6] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional Attention Flow for Machine Comprehension. *arXiv:1611.01603 [cs]*, Nov. 2016. arXiv: 1611.01603.

Appendix

Graphs obtained from selected runs