Question Answering on the SQuAD Dataset

Yongshang Wu, Hao Wang
Department of Computer Science
Stanford University
{wuy, wanghao}@stanford.edu

Abstract

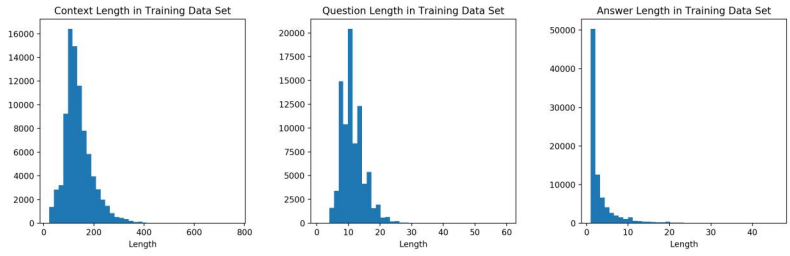
Question answering (QA) is a challenging task in natural language processing, where a complex modeling of the context and the question is necessary for high quality prediction. In this project, we tackled the QA task on the SQuAD dataset. We implemented and refined various techniques including char-level embedding, gated dot attention, self-attention, multiplicative attention, pointer networks, span selection with dynamic programming as well as ensemble models. As a result, the final ensemble model achieved 77.89 F1 and 68.37 EM on the test set. The visualization and error analysis further explained the effectiveness of our model.

1 Introduction

Question answering is a task where a system is required to predict a answer given a context and a query. Such systems have many practical applications and the task is still a challenging topic in the field of natural language processing. In order to obtain high quality prediction, a system should model the complex relation between the context and the question by learning a large amount of data.

Stanford Question Answering Dataset (SQuAD) [1] is a recent question answering dataset with more than 100,000 question-answer pairs and 500 articles. This project tackled the QA task on the SQuAD dataset. On the basis of a baseline model, we implemented improvements across all layers, including char-level embedding, gated dot attention, self-attention, multiplicative attention, pointer networks, span selection with dynamic programming as well as ensemble models. Our final ensemble model achieved 77.89 F1 and 68.37 EM on the test set. We further conducted attention visualization and error analysis to explained the effectiveness of our model.

2 Dataset Analysis



(a) (b) (c)

Figure 1: Length of (a) context, (b) question and (c) answer in the training dataset.

In order to get insights from the dataset, we analyzed the length of context, question and answer before actual training (Figure 1). The histogram of length of contexts show that most of contexts are shorter than 450 words. This discovery inspired us to limit the maximum context length of our models, which reduced training time.

3 Method

3.1 Architecture Overview

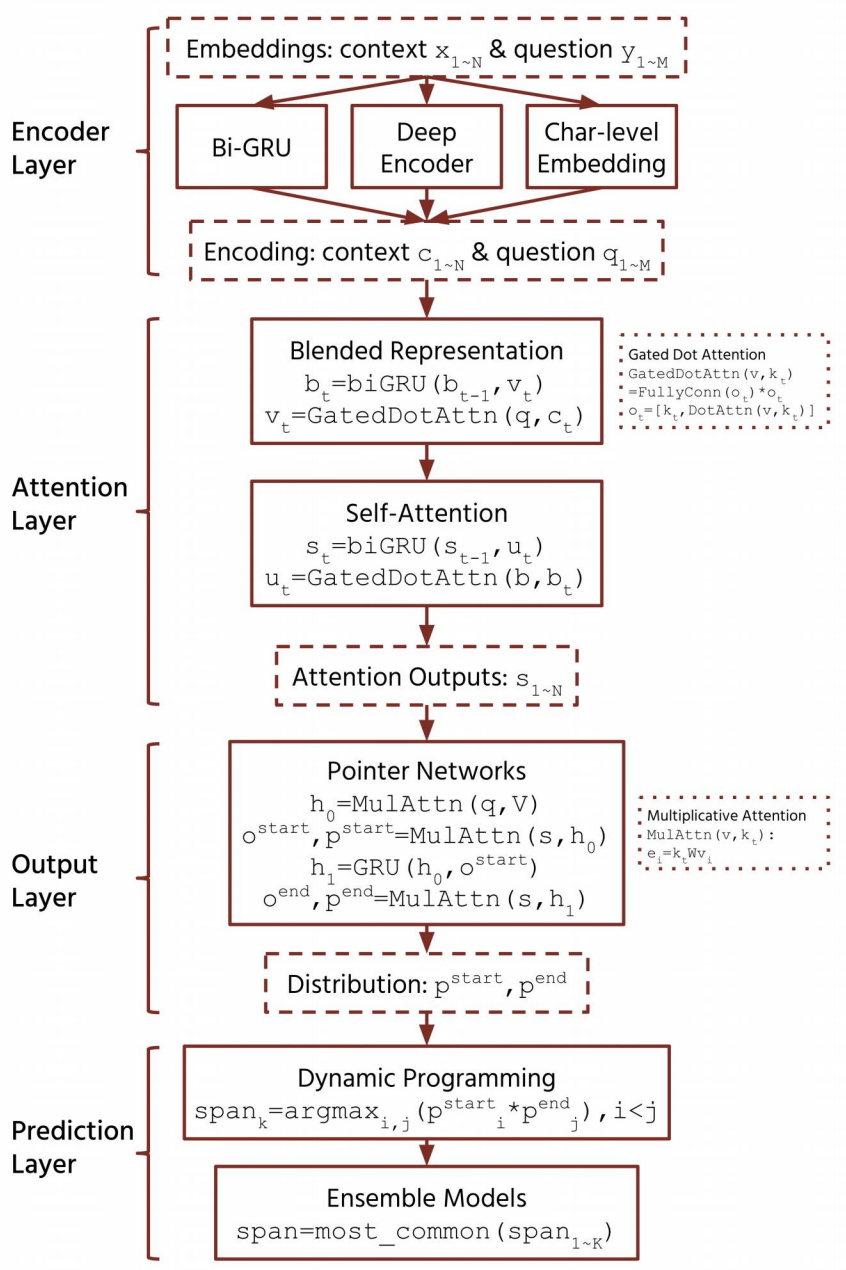


Figure 2: Architecture overview.

Based on a baseline model, we experimented various technique. Figure 2 summarized all techniques we implemented, which can be organized into four layers: encoder layer, attention layer, output layer

and prediction layer. In the encoder layer, apart from the basic bidirectional GRU units, we tried bi-GRU with deeper layers as well as character-level embeddings. In the attention layer, the simple dot-product attention was replaced with a gated dot attention followed by self-attention. Pointer networks were adopted in the output layer to condition end prediction on start prediction. In the prediction layer, smart span selection with dynamic programming as well as ensemble models were used to further improve performance of our models.

3.2 Encoder Layer

For each SQuAD example, the context is represented by a sequence of word embeddings $x_1, x_2, \ldots, x_N \in \mathbb{R}^d$, and the question by a sequence of word embeddings $y_1, y_2, \ldots, y_M \in \mathbb{R}^d$. These embeddings are then fed into an encoder layer to produce new representations $c_1, c_2, \ldots, c_N \in \mathbb{R}^{2h}$, $q_1, q_2, \ldots, q_M \in \mathbb{R}^{2h}$ that could integrate richer information of the context they are in, where h is the hidden size of all the RNNs we use in our implementation. In this section, different types of encoder layers experimented in our implementation are disccussed.

3.2.1 Bidirectional GRU

For bidirectional GRU [2] encoder, embeddings are fed into a shared-weight 1-layer bidirectional gated recurrent unit network:

$$c_i = \text{biGRU}(c_{i-1}, x_i)$$
$$q_i = \text{biGRU}(q_{i-1}, y_i)$$

We simply concatenate the forward and backward hidden states to obtain the encoding of context and question.

3.2.2 Deep Encoder

Based on bi-GRU encoder, we could stack more layers into the GRU to generate more complex encodings.

$$c_i^{[l]} = \text{biGRU}(c_{i-1}^{[l]}, c_i^{[l-1]}, x_i)$$
$$q_i^{[l]} = \text{biGRU}(q_{i-1}^{[l]}, c_i^{[l-1]}, y_i)$$
$$\text{for } l = 1, 2, 3$$

We use the depth-concatenated forward and backward hidden states to obtain the encoding of context and question:

$$c_i = [c_i^{[3]}, c_i^{[2]}, c_i^{[1]}]$$
$$q_i = [q_i^{[3]}, q_i^{[2]}, q_i^{[1]}]$$

To prevent overfitting due to a deeper model, we also apply dropout on the deep encoder layer.

3.2.3 Character-level Embedding

Since the vocabulary size of embedding matrix is limited, out-of-vocabulary (OOV) words are often encountered. Instead of using fixed OOV embedding to represent these words, we could use pretrained character-level embedding in addition to word embeddings to gain more expressive representation. Following [3], we feed characters char_{i1}^c, char_{i2}^c, ..., char_{ij}^c of the i-th word in context and char_{i1}^q, char_{i2}^q, ..., char_{ij}^q of the i-th word in question to a bidirectional GRU:

$$ch_i^c = \text{biGRU}(ch_{i-1}^c, \text{char}_i^c)$$
$$ch_i^q = \text{biGRU}(ch_{i-1}^q, \text{char}_i^q)$$

We take the final hidden states of the bi-GRU as the character-level embedding of a word and concatenate it to the word embedding:

$$x_i = [x_i, ch_i^c]$$
$$y_i = [y_i, ch_i^q]$$

before feeding embeddings into encoder layers mentioned in 3.2.1 and 3.2.2.

3.3 Attention Layer

3.3.1 Blended Representations

[3] proposed a gated attention-based recurrent network. [4] proposed generating sentence-pair representation via soft-alignment of words in context and question using RNN. Similarly, we implemented the context-to-question attention using gated attention. But in lieu of incorporating hidden states in attention computing, we simply concatenate the attention output to the context encoding and feed the blended representation into a bidirectional GRU. This simplification allows us to use simple dot-product instead of additive attention used in [4] where there are two attention targets. Such modification decreases the number of parameters drastically and reduces a significant amount of computation. In a dot-product attention DotAttn(v, k), attention score $e_i \in \mathbb{R}$ for each value $v_i \in \mathbb{R}^d$ with respect to key $k_t \in \mathbb{R}^d$ is calculated as follows:

$$e_i = k_t^\intercal v_i$$

In a gated dot-product attention GatedDotAttn(v, k), the dot-product attention DotAttn(v, k) is first computed, then the attention output for the t-th key, GatedDotAttn(v, k_t), is concatenated to the key itself:

$$o_t = [k_t, \text{DotAttn}(v, k_t)]$$

A fully connected layer with ReLu activation is afterwards applied on o_t to compute the gate for this particular output, the final output is the element-wise product of this gate and the blended output:

$$\text{GatedDotAttn}(v, k_t) = \text{FullyConn}(o_t) * o_t$$

Given the context representation set c and question representation set q, we first compute the gated dot-product attention:

$$\text{v} = \text{GatedDotAttn}(q, c)$$

Then we feed the attention output into a bidirectional GRU:

$$b_t = \text{biGRU}(b_{t-1}, v_t)$$

The forward and backward hidden states are concatenated to obtain the final blended representation.

3.3.2 Self-attention

[3] proposed directly matching the question-aware passage representation against itself to dynamically collect and encode evidence from the whole passage into the passage representation. As in 3.3.1, the gated dot-product attention is used to match the blended passage representation to itself:

$$\text{u} = \text{GatedDotAttn}(b, b)$$

Again, the attention output is fed into a bidirectional GRU:

$$s_t = \text{biGRU}(s_{t-1}, u_t)$$

and the final self-aware passage representation is obtained by concatenating forward and backward hidden states.

3.4 Output Layer

3.4.1 Pointer Networks with Multiplicative Attention

We followed [5] and [3] and adopted pointer networks to predict probability distributions for the start and end position of the answer. Similar to [3], we used an attention-pooling on the question representation q to generate a initial hidden state for the pointer network. However, unlike its original implementation, our version used multiplicative attention instead of additive attention. This change reduced the number of parameters in the models while kept flexibility of dimensions of input matrices. In a multiplicative attention MulAttn(v, k), attention scores for the value is defined as follows:

$$e_i = k_t W v_i,$$

where $k_t \in \mathbb{R}^{d_1}$ is a key, $v_i \in \mathbb{R}^{d_2}$ is a value and $W \in \mathbb{R}^{d_1 \times d_2}$ is a weight matrix.

Given the question representation q and a random initialized parameter vector $V \in \mathbb{R}^{2h}$, the pointer network first computes a initial hidden state $h_0 \in \mathbb{R}^{2h}$:

$$h_0 = \text{MulAttn(q,V)}.$$

Then a multiplicative attention is used to predict the probability distribution for the start position:

$$o^{start}, p^{start} = \text{MulAttn}(s, h_0),$$

where the initial hidden state h_0 attends to output of the attention layer s and compute the probability distribution p^{start} as well as the output o^{start}.

Next, a GRU cell is employed to compute a new hidden state $h_1 \in \mathbb{R}^{2h}$:

$$h_1 = \text{GRU}(h_0, o^{start}).$$

Finally, by attending h_1 to s again, the probability distribution p^{end} as well as the output o^{end} of the end position:

$$o^{end}, p^{end} = \text{MulAttn}(s, h_1).$$

With the hidden state h_1, we are able to conditioning end prediction on start prediction. Furthermore, if the number of position is more than 2, we can simply add more hidden states in the GRU.

3.5 Prediction Layer

The prediction layer computes final predicted spans (l^{start}, l^{end}) from p^{start} and p^{end}. The baseline model simply takes the argmax over probability distributions to obtain predicted spans. In this project, we improved the prediction layer with smart span selection and ensemble models.

3.5.1 Span Selection with Dynamic Programming

A straightforward improvement of the baseline prediction layer is conditioning $l^{start} < l^{end}$, e.g.,

$$l^{start}, l^{end} = \arg\max_{i,j}(p_i^{start} * p_j^{end}), i < j.$$

In order to implement this method efficiently, we adopted a dynamic programming algorithm. Specifically, we first compute $p^{start,max}$ as a cumulative max of p^{start}. Then the max probability product is found by:

$$\max_j(p_j^{start,max} * p_j^{end}).$$

In this way, the conditioned span selection can be computed in linear time $O(N)$.

3.5.2 Ensemble Models

We further implemented ensemble models, which is a common strategy for improving performance. For the same context and question, the final span is the most common one in predicted spans from K models.

4 Results and Analysis

4.1 Performance

#	Model	Embedding	Dataset	F1	EM
1	Baseline	100d	Local Dev	40.13	29.10
2	Char-level Embedding	100d	Local Dev	40.19	29.47
3	Deep Encoder	100d	Local Dev	41.15	30.71
4	Pointer Network	100d	Local Dev	43.62	30.56
5	Self-Attention	300d	Local Dev	70.87	60.42
6	Self-Attention + Pointer Network	300d	Local Dev	72.08	61.49
7	Self-Attention + PtrNet + DP	300d	CodaLab Test	74.32	63.59
8	**Self-Attention + PtrNet + DP + 10 Ensemble**	**300d**	**CodaLab Test**	**77.89**	**68.37**

Table 1: Performance of experiments.

Table 1 shows the experiments we conducted. Experiment 1 is the baseline model using a shared-weight 1-layer encoder over 100-dimensional word embedding, vanilla dot-product context-to-question attention and softmax output layer for predicting start and end position. Experiments 2, 3, 4 and 5 only change single control variable (one component) of the baseline model, as stated by thier names. As we can see, self-attention boosts the performance on F1 and EM scores most drastically. For experiment 6 and 7, we builds on the self-attention-based model by adding pointer network and span selection with dynamic programming. At last, we run the best model (with self attention, pointer network and DP-based span selection) with 10 different random seeds and combine them into an ensemble model. The final CodaLab test leaderboard submission reports XXX F1 and XXX EM. The changes we made on the hyperparameters when training best model are: context_len from 600 to 450 and embedding_size from 100 to 300, while others remain default.

4.2 Attention Visualization

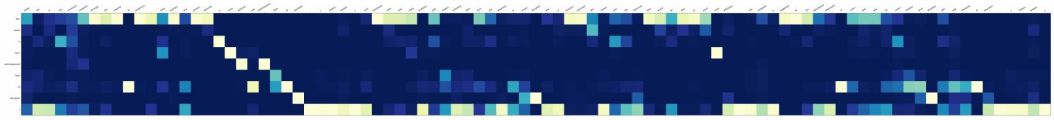


Figure 3: Visualization of gated dot attention.

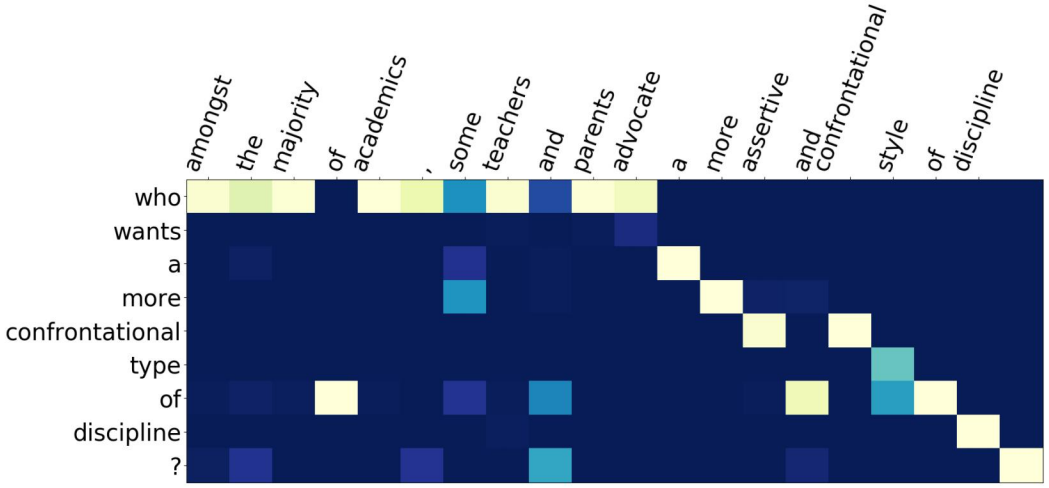


Figure 4: Visualization of gated dot attention.

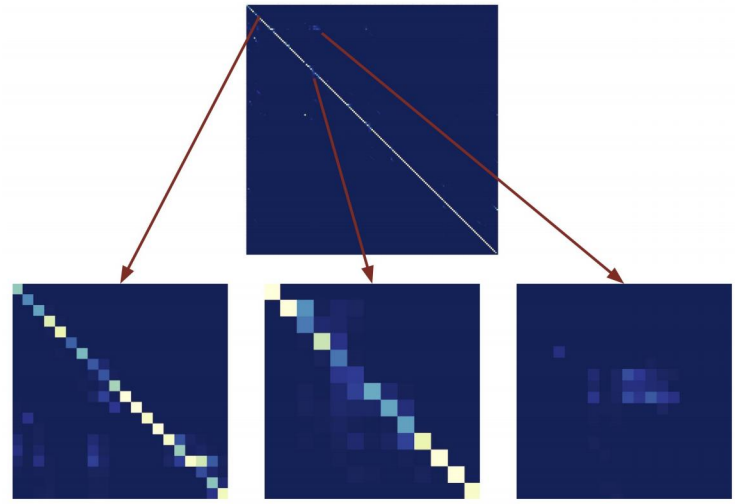


Figure 5: Visualization of self-attention.

Figure 3 is the visualization of gated dot-product context-to-question attention distribution on an example, where the x-axis is the context and the y-axis is the question. In Figure 4 we zoom in to the answer span for details of the distribution. As we can see, the answer "teachers and parents" achieve much higher attention scores for the question word "who", and same words or similar words (e.g. "confrontational" and "assertive") tend to have higher attention scores. This means the gated dot attention could somehow discover and highlight the correlation of context and question.
Figure 5 is the visualization of self-attention distribution over an example. It is a bit hard to interpret this relatively high-level attention, but by zooming into parts of the matrix we could still see that the attention mechanism allows question-aware context to encode information of neighboring and/or related representation.

4.3 Error Analysis

We pick several examples where our model failed, to gain a sense of how it may be further improved.

4.3.1 Example 1

- **Context**: before the st. elizabeth 's flood (1421) , the meuse flowed just south of today 's line merwede-oude maas to the north sea and formed an archipelago-like estuary with waal and lek . this system of numerous bays , estuary-like extended rivers , many islands and constant changes of the coastline , is hard to imagine today . from 1421 to 1904 , the meuse and waal merged further upstream at gorinchem to form merwede . for flood protection reasons , the meuse was separated from the waal through a lock and diverted into a new outlet called " bergse maas " , then amer and then flows into the former bay hollands diep .
- **Question**: where did the meuse flow before the flood ?
- **Prediction**: bay hollands diep
- **Answer**: merwede-oude maas

The question is where did the meuse flow **before** the flood but our model gives where it flowed **afterwards**. This mistake may be caused by the high correlation between the words before our prediction and the question, and this happens to occur at the end of the context while the true label is at the beginning. So it is difficult for GRU to learn proper gate control to choose the early span not the one at the end. Switching to LSTM with more complicated gates may be helpful.

4.3.2 Example 2

- **Context**: the centre-left australian labor party (alp) , the centre-right liberal party of australia , the rural-based national party of australia , and the environmentalist australian

greens are victoria 's main political parties . traditionally , labor is strongest in melbourne 's working class western and northern suburbs , and the regional cities of ballarat , bendigo and geelong . the liberals ' main support lies in melbourne 's more affluent eastern and outer suburbs , and some rural and regional centres . the nationals are strongest in victoria 's north western and eastern rural regional areas . the greens , who won their first lower house seats in 2014 , are strongest in inner melbourne .

- **Question**: what party is favored in bedigo and geelong ?
- **Prediction**: centre-left australian labor party
- **Answer**: labor

The prediction is correct to some extent since it selects out a reasonable answer span, which is not too large and contains the true label. Adding POS features and analyzing syntactic structures of prediction may help generate more precise answers.

5 Conclusion

In this paper, we implemented various techniques on the task of question answering in the SQuAD dataset. The results showed the ensemble model with self-attention, pointer networks and span selection with dynamic programming achieved high performance, comparing to the simple baseline model. The analysis on attention visualization indicated the gated dot-product context-to-question attention learned correlation between the context and the question, and the self-attention allowed the question-aware context to encode neighboring information. One future work can be looking into the minor improvement from the character-level embedding and the deep encoder. Another is adding POS features and analyzing syntactic structures of prediction.

Acknowledgments

We would like to thank Richard Socher and all CAs for the excellent learning experience. We also acknowledge Microsoft for providing GPU virtual machines on Azure and inspiring us with R-Net.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

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

[4] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*, 2015.

[5] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016.