Machine Reading Comprehension on SQuAD with Relevance Encoder

Feng Liu, Qixiang Zhang
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
{liufeng, qixiang}@stanford.edu

Abstract

The aim of this study is to build an artificial intelligence system with deep neural networks to accomplish machine reading comprehension tasks. This paper reports the development of new models and details of experiments with the Stanford Question Answering Dataset (SQuAD). The task requires models to answer certain questions using a sequence of words extracted consecutively from a given passage of context. A new model, the Relevance Encoder, which achieved F1 score of 71.9% and EM score of 61.3% on the CodaLab Test Leaderboard, significantly boosted the performance of the baseline model with bidirectional LSTMs and dot-product attention. Fully connected output layer, Answer Pointer, and RNN output layer were all implemented and compared. Span search is used to determine the final output. Details of each network architecture, performance, and training speed are provided for analysis and comparison.

1 Introduction

Reading comprehension has long been a challenge for both human beings and machines. The performance of machines in completing question answering tasks to a certain degree indicates the level of development of artificial intelligence. Several criteria have been developed for evaluating the performance of the models. Generally, the forms of the questions and answers may vary in a range, and depend on the dataset that is adopted in training. As is observed in many other artificial intelligence challenges, the performance of the networks is largely influenced by the quality and size of the dataset used for training.

The Stanford Question Answering Dataset (SQuAD) by Rajpurkar et al. [1] has become one of the most popular datasets for machine reading comprehension research since its appearance in 2016. SQuAD provides more than 100,000 question-answer pairs on more than 500 wikipedia articles labeled by crowdworkers as training examples. In the training and evaluating processes, passages and associated questions are used as input to the model, and consecutive subsequences (one for each question) extracted from the original passages are used as labels.

End-to-end learning has been considered as an prevailing advantage of deep neural networks. On on hand, compared to traditional methods, end-to-end neural networks save the effort of linguistic analysis and feature extraction in preprocessing texts. Many successful models are built based on this methodology and have made significant breakthroughs. Models such as Match-LSTM by Wang et al. [2] and Bidirectional Attention Flow (BiDAF) by Seo et al. [3] are paradigms of end-to-end learning. On the other hand, a proper amount of feature engineering in return may help in reducing the complexity of networks and increasing the training capacity without loss of accuracy. DrQA by Chen et al. [4] explores possibilities in this domain, namely exact match, token features, and aligned question embedding, and achieved comparable performance without complex attention layers or even RNNs following.

In this project we consider conducting some preliminary experiments in balancing between these two types of natural language processing methodologies.

2 Method

In the following we describe the Relevance Encoder model. The basic idea is that, though our Relevance Encoder model does not aim to make heavy use of multilayer RNNs, the output of each established RNN layer should be exploited. The structure of our Relevance Encoder model is similar to the baseline model that is assembled with an encoder layer, a attention layer, and an output predictor layer.

2.1 Encoder Layer

The encoder layer is composed of a bidirectional LSTM encoder for questions, which encodes the questions into hidden state vectors; the Relevance Encoder for context with questions, which concatenates raw context word vectors with relevant question hidden state vectors; and one another following LSTM encoder for context, which encodes the relevance-encoded context in to hidden state vectors of the same size as the questions.

2.1.1 Question Encoder

The question encoder used here is a simple bidirectional LSTM layer with hidden size h in each direction.

$$q = \text{biLSTM}(e_q) \in \mathbb{R}^{M \times 2h}$$

where e_q is the raw word embedding vectors of the question and M is the length of the question.

2.1.2 Relevance Encoder

In the DrQA, the embedding of question words does not pass through RNNs before it is used to compute the similarity score with context words which come though an RNN encoder. Considering that questions are relatively short, or usually they can be truncated, it will not slow down unacceptably the computation if we encode questions into hidden vectors with bidirectional LSTM, which may help to enrich the information carried by each word in a sentence by taking word order into account. Now that questions have been encoded, it is natural to weighted sum over question hidden vectors instead of raw question word embedding vectors as an extra feature. The weights for this step of averaging are calculated with raw word embedding vectors of context passages and questions.

This step is similar to the basic dot-product attention, while the values are treated differently. The i-th key used here is the raw embedding vector of the i-th word in the context $e_c^i \in \mathbb{R}^d$. Each value is split into two parts: $[q_j; e_q^j] \in \mathbb{R}^{2h+d}$. For each context word e_c^i, the attention distribution $a^i \in \mathbb{R}^M$ is obtained as:

$$\alpha^i = {e_c^i}^T [e_q^1, ..., e_q^M] \in \mathbb{R}^M$$
$$a^i = \text{softmax}(\alpha^i) \in \mathbb{R}^M$$

The relevance feature r^i for the i-th word in the context is further calculated by taking weighted sum of question hidden vectors with weight vector a^i:

$$r^i = \sum_{j=1}^{M} a_j^i q_j \in \mathbb{R}^{2h}$$

The blended context word vectors are then yielded by concatenating e_c and r:

$$\tilde{c}_i = [e_c^i, r_i] \in \mathbb{R}^{d+2h}$$

2.1.3 Context Encoder

The context encoder is a bidirectional LSTM that takes relevance encoded context word vector $\tilde{c}$ as input and transform it into the same format as question.

$$c = \text{biLSTM}(\tilde{c}) \in \mathbb{R}^{N \times 2h}$$

Note that the question encoder and the context encoder do not share weights; they are not of the same size.

2.2 Attention Layer

Three types of attention mechanisms are implemented and compared in this paper, namely basic dot-product attention, matrix product attention, and Bidirectional Attention Flow (BiDAF). All three mechanisms are to be introduced below, of which the one with the best performance is adopted in the final model. Analysis is included in the *Discussion* section.

2.2.1 Basic Dot-Product Attention

Basic dot-product attention is used in the baseline model.

The i-th key used here is the hidden vector of the i-th word in the context $c_i \in \mathbb{R}^{2h}$. The j-th value used here is the hidden vector of the j-th word in the question $q_j \in \mathbb{R}^{2h}$. The attention distribution $a^i \in \mathbb{R}^M$ is obtained as:

$$\alpha_i = {c_i}^T [q_1, ..., q_M] \in \mathbb{R}^M$$
$$a_i = \text{softmax}(\alpha_i) \in \mathbb{R}^M$$

The relevance feature r^i for the i-th word in the context is further calculated by taking weighted sum of question hidden vectors with weight vector a_i:

$$r_i = \sum_{j=1}^{M} a_{ij} q_j \in \mathbb{R}^{2h}$$

The blended representation for each context word is then yielded by concatenating c_i and r_i:

$$x_i = [c_i, r_i] \in \mathbb{R}^{d+2h}$$

2.2.2 Matrix Product Attention

Similar to the basic dot-product attention described above, matrix inner product attention introduces a matrix $W \in \mathbb{R}^{2h \times 2h}$ in the product. The attention score between c_i and q_j is hence:

$$a^{ij} = c_i^T W q_j$$

The remaining portion of the layer remains the same.

The intuition to introduce this matrix W is to align the hidden vectors of the context and question words, since they are not outcome of the same RNN encoder.

2.2.3 BiDAF

In the previous attention layers, only the context words attend to question words, not the reverse direction. As an extension, BiDAF allows attention to flow in both directions.

The mathematical expression is the same as in the default project guide section 5.1.1.

2.3 Output Layer

The output of the attention layer is fed into the output layer to make the prediction of answer's start and end positions. There are different methods in constructing this function.

2.3.1 Fully Connected Layer

Using a fully connected layer is simple. Feed the attention output $x \in \mathbb{R}^{N \times D_x}$ into a fully connected layer to get the logits for each word to be the start or end of the answer:

$$\hat{y}_{\text{start}} = f(W_{\text{start}} x + b_{\text{start}})$$
$$\hat{y}_{\text{end}} = f(W_{\text{end}} x + b_{\text{end}})$$

Here $f(\cdot)$ is the activation function.

Then $\hat{y} \in \mathbb{R}^N$ is used to calculate loss.

2.3.2 Answer Pointer

This module is to depend end position on start position. The start positions are predicted with a start weight matrix. The end positions are predicted with by an end weight matrix toghether with the result of start predictions.

2.3.3 RNN Layer

The attention output is a sequence with length N. The order of words is till of great importance. Therefore, RNN layer is proposed to be used here as

$$\hat{y}_{\text{start}} = \text{biRNN}_{\text{start}}(x)$$
$$\hat{y}_{\text{end}} = \text{biRNN}_{\text{end}}(x)$$

The remaining portion is the same as fully connecty output layer.

2.3.4 Span Search Trick

When making the final prediction, a trick is applied here. The predictions of start and end positions is selected to maximize the product of $p^i_{\text{start}} p^j_{\text{end}}$ with $0\ leq j - i \leq 15$. The span 15 is obtained from answer length distribution shown in Figure 1.

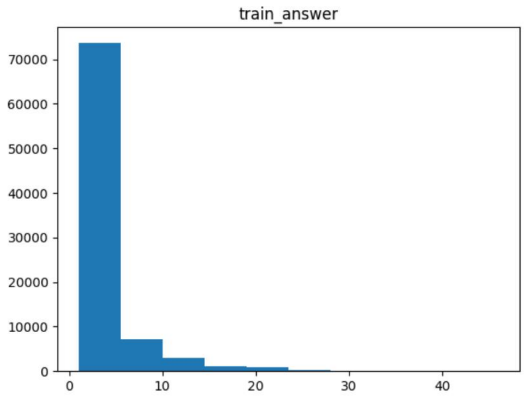


Figure 1: Answer Length Distribution in Training Set

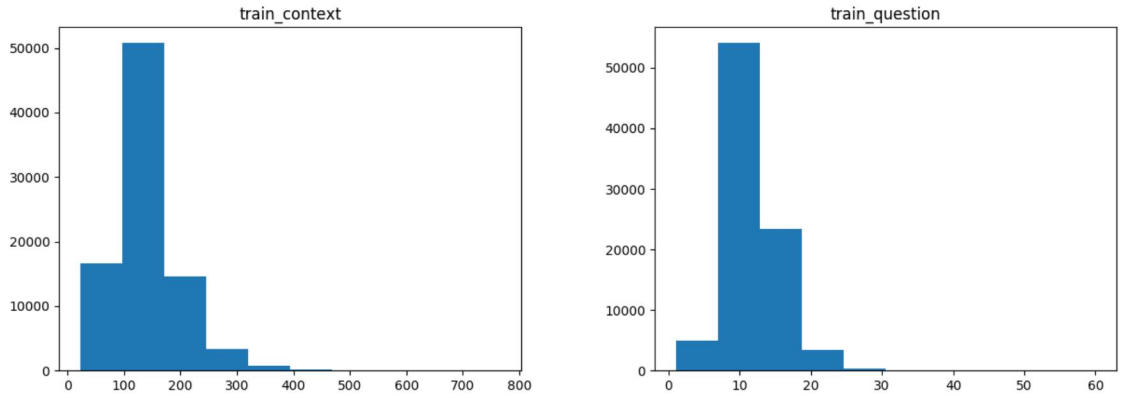
3 Experiments

In this section, we provides details of experiments, including data, settings, and results.

3.1 Data

The dataset we use is SQuAD. The training set contain 86,318 triplets of (context, question; answer); the dev set contains 10,391.

For computing efficiency, both contexts and questions are padded or truncated to a proper length. Figure 2 shows the distribution of text length in training set.



(a) (b)

Figure 2: Context and Question Length in Training Set

Context length of 600 and question length of 30 appears to be a safe starting point. By further testing, we found using context length of 400 and question length of 30 does not impact severely on performance, with a drop of F1 score of approximately 0.3%.

3.2 Experiment Settings

The word embedding used in this experiment is the GloVe vectors pre-trained with 6 billion Wikipedia and Gigaword of vocabulary size of 400,000. The word vector dimension finally used in our best model is 100. We use this number because previous experiments of Ke et al.[5] show that increasing word vector dimension from 100 to 200 does not improve performance significantly.

We first select potential models with priliminary hyper-parameters. In this step, we use hidden layer dimension of 200 for all RNNs used in th networks. And we use the default setting of Adam optimizer wiht learning rate 0.007. The dropout rate is set as 0.15. No norm regularizer is used.

Both Exact Match and F1 scores are tracked during training processes.

3.3 Results

3.3.1 Models Comparison

The primary results listed in Table 1 are preliminary results used for selecting a potential model to explore further. When comparing training speed, batch size of 1000 is used. These experiments were early stopped if the trends are clear enough to make the selection. The Relevance Encoder appears to be a promising model together with the LSTM output layer. Earlier research papers [3][5] report high performance of BiDAF implementations. However, when BiDAF is implemented together with Relevance Encoder, the best dev scores do not increase, with only the training time almost doubles. Therefore, we decided to move on with Relevance Encoder and basic attention.

Table 1: Priliminary Model Selection

Encoder	Attention	Output	Params Number	Exact Match Dev	F1 Dev	Time / 1k iter
LSTM	Basic	FC	0.6M	30.64	41.83	23m
LSTM	BiDAF	FC	0.8M	35.46	47.77	35m
LSTM	BiDAF	Pointer	1.0M	36.37	48.98	34m
Relevance	Basic	FC	1.7M	46.55	61.21	27m
Relevance	Basic	Pointer	1.9M	45.74	60.72	43m
Relevance	Matrix	FC	1.9M	46.58	61.14	27m
Relevance	BiDAF	FC	1.9M	46.51	60.90	43m
Relevance	Basic	LSTM	3.2M	49.10	63.63	47m
Relevance	Basic	LSTM + Span15	3.2M	**51.70**	**66.18**	50m

3.4 Best Results

After some parameter tuning with best model listed in the table above (Relvence Encoder + Basic Attention + LSTM output + Span Search), we obtained the final dev and test scores in Table 2.

Table 2: Best Model Performance

			Exact Match		F1	
Encoder	Attention	Output	Dev	Test	Dev	Test
Relevance	Basic	LSTM + Span15	**61.523**	**61.345**	**71.908**	**71.882**

4 Discussion and Conclusion

4.1 Error Analysis

Figure 3 below is a histogram comparing length difference between ground truth answer sequence and predicted answer sequence. An example is counted only when sequence lengths are different (i.e. len(truth) != len(prediction)).

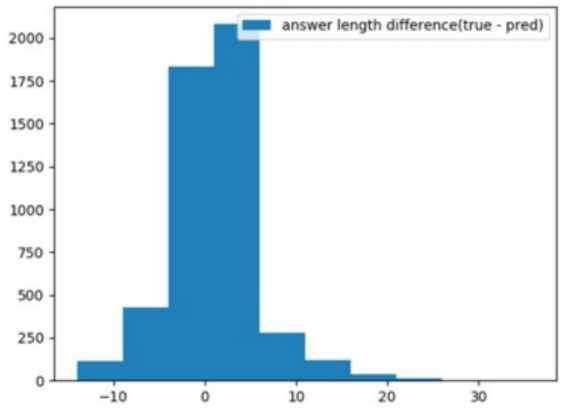
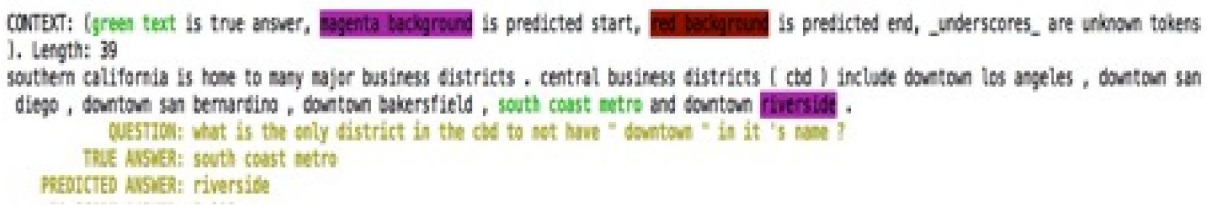
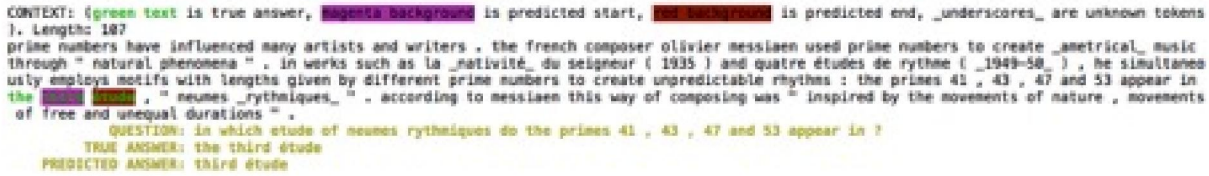


Figure 3: Difference of Length Between Predicted and True Answer

From the histogram we can see that, the model tends to predict overly short answers. Below is an example:



From the example we can conclude that sometimes the start index is already close to end of context, such that end index is assigned to be the end of index. This is a suggests effect of our hard assignment of start/end indices (part 5.7). Here is another example:



In this example, the predicted answer “third étude” is correct in some sense, but the true answer “the third étude” leads to impreciseness of the prediction. This suggests that articles are neglected by the model at times.

4.2 Discussion

Our experiments for model comparison and selection are not exhaustive. It is possible to boost certain model by setting a exclusively suitable set of hyper-parameters. However, to save both time and resources, greedy method is adopted to a certain degree.

Matrix product attention does significantly improve the performance of Relevance Encoder compared to basic dot-product attention. As is explained in the *Method* section, the motive to introduce matrix attention is to align the hidden vectors of context and question words because they come from different RNN encoders. This result can be explained as, this similarity requirement can be learned in the end-to-end manner. For the same raison, contexts and questions are not necessarily fed into the same RNN with weights shared, because the network may be able to learn whatever needs to be aligned with other nuances remaining.

We observe that appending BiDAF after Relevance Encoder does not improve the performance of Relevance Encoder, but slows down the training process significantly. This observation suggests that the Relevance Encoder in some extent could replace BiDAF and accelerate training. Additionally, in experiments with all other settings same, we find that Relevance Encoder alone outperforms BiDAF alone in the priliminary experiments. Our BiDAF model is not optimal, with only one RNN encoder before. However, Relevance Encoder with basic attention is a simpler model using additioanl features and less RNN layers. Therefore, we stopped exploring BiDAF because of the consideration of its time consumption. An additional issue is that BiDAF requires heavy use of memory and easily causes resource exhaustion. To circumvent this problem, slicing and concatenation are done before and after huge matrix operations to keep the batch size equal. This tradeoff for memory usage makes BiDAF even slowlier.

What might matter here is the number of times context coupled with question is passed through different layers of RNNs. By comparing with successful BiDAF papers such as [5], we find that their BiDAF is followed by one or two additional layers of LSTM. These following RNNs may work in a similar way to the RNN output layer in our model. In our best model, the context passes RNN two times: once in relevance encoder, another time in the output layer. Both times it is encoded together with question information.

4.3 Conclusion

In this paper we report the results of our Relevance Encoder model, the details of implementation, and the comparison with other models. In experiments, the Relevance Encoder achieved best F1 score of 71.9% and EM score of 61.3% on the CodaLab Test Leaderboard. This model adopts the relevance of raw text similarity as additional features, and shows that feature engineering is potential to achieve comparable performance without RNNs stacking. There is still space to improve the model by further combining more complex attention mechanisms, increasing the number of RNN layers, introducing token-level or character-level features, etc.

Acknowledgments

This research is a part of Stanford CS224N course project, and is supported by the teaching staff and the department. The computing resources are provided by Microsoft Azure.

References

This research is a part of Stanford CS224N course project, and is supported by the teaching staff and the department. The computing resources are provided by Microsoft Azure.

[1] Rajpurkar, P. et al., (2016) Squad: 100,000+ Questions for Machine Comprehension of Text. arXiv:1606.05250.

[2] Wang, S. et al., (2016) Machine Comprehension Using Match-LSTM and Answer Pointer. arXiv:1608.07905.

[3] Seo, M. et al., (2017) Bi-directional Attention Flow for Machine Comprehension. arXiv:1611.01603.

[4] Chen, D. et al., (2017) Reading Wikipedia to Answer Open-Domain Questions. arXiv:1704.00051.

[5] Ke, J. et al., (2017) Question Answering System with Bi-Directional Attention Flow.