Multi-Perspective Context Matching for SQuAD Dataset

Huizi Mao
Stanford University
huizi@stanford.edu

Xingyu Liu
Stanford University
xyl@stanford.edu

Abstract

Question answering is an important task in machine comprehension. The new SQuAD dataset allows us to deploy recent NLP deep learning techniques and train an end-to-end system to predict the start and end position of the answer in the given context, instead of precisely selecting the words of the correct answer. We propose to use combine bi-directional LSTM (BiLSTM) and context matching to develop a model for SQuAD dataset. We first use two BiLSTMs to encode the context and question word sequence. Then we apply context matching from multiple perspectives to produce a matching vector. Finally another BiLSTM is applied to the matching vector to predict the start and end positions. Several other tricks are also explored to enhance the prediction accuracy. Experimental results show that our model can achieve an F1 score of 61.27 and EM score of 49.50 on the development set.

1 Introduction

The newly released Stanford Question Answering dataset (SQuAD) [2] is a large manually labeled dataset with over 100,000 question-answer pairs on over 500 articles. Compared to previous dataset, SQuAD's great capacity of both the size and diverse question-answer style enables us to develop NLP deep learning model based on it. We referred to to implement our own model.

Our model consists of three parts: 1) the first two BiLSTMs encode the context and question word sequence into a vector; 2) context matching from multiple perspectives produces matching vector; 3) the third BiLSTM with a Softmax loss on the matching vector predicts the start and end positions of the answer. We also examined several optimization tricks that improved the accuracy on development set.

The following part of the paper is organized as follows: in Section 2, we describes approaches we used, including the architecture of our deep model and the preprocessing techniques; in Section 3, we report the experiment result; in Section 4, we describe the observations from our experiment and discuss the possible tricks to improve the accuracy.

2 Approach

Based on the architecture and training flow in , we implemented our own model. The architecture of the model is illustrated in Fig. 1. The following subsections will explain each part of the training flow in detail.

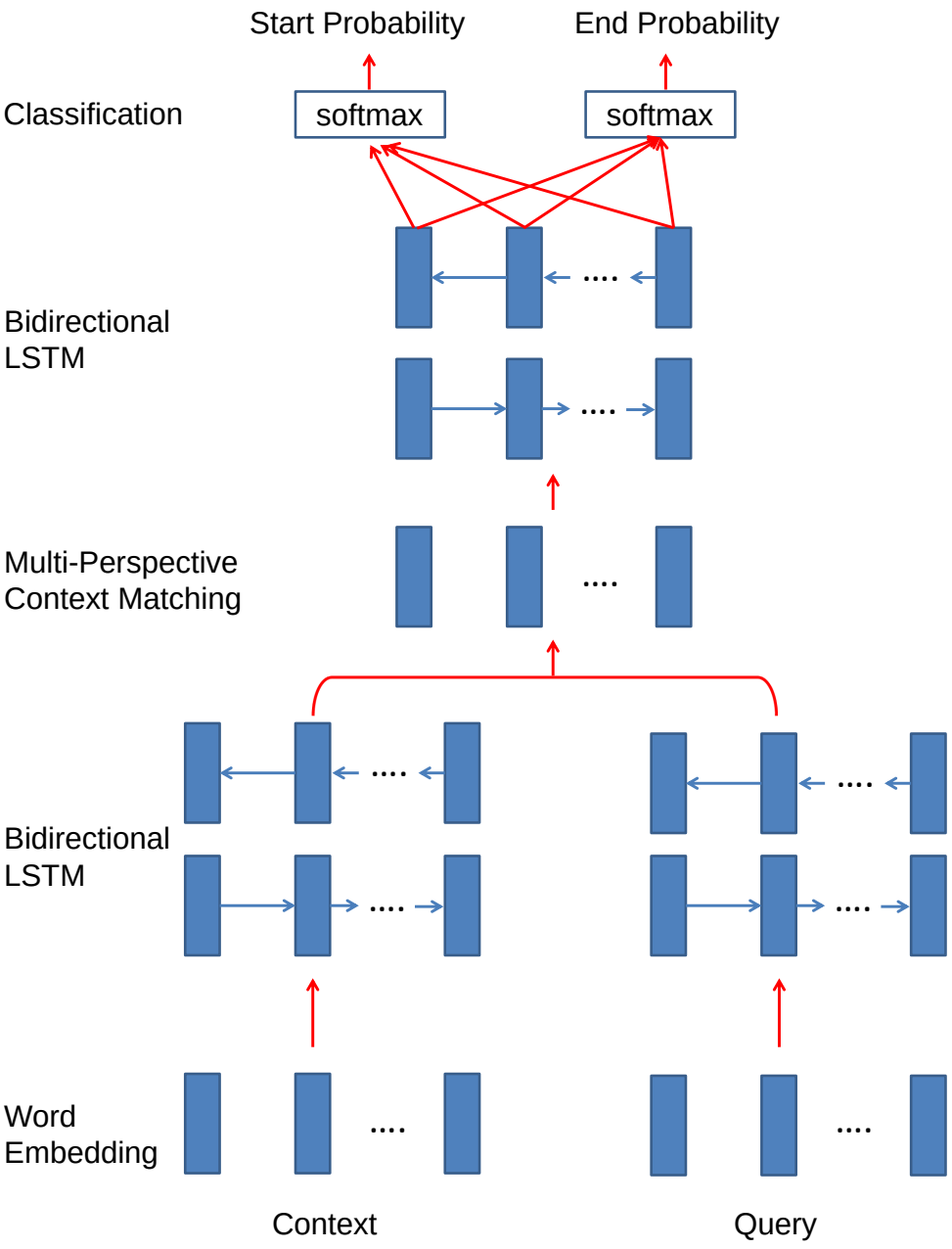

Figure 1: Model architecture

2.1 Preprocessing

The first stage of our training is the word representation. This stage convert the every word in the context and question into a vector with a length of d. Due to the limit of time, we only used the word embedding instead of character-composed embedding. We used provided GloVe as the pre-trained word embedding. For the characters not included in GloVe, we mapped them to a fixed token. The tokenized words are then trained to be mapped to a d-dimensional vector. The mapping is initialized randomly and will be trained during training time. We used $d = 100$ in this project. Other d sizes might also work and the design space exploration will be left as future work.

2.2 Word Encoding

The converted d-dimensional word vectors from the paragraph context of length m and the question of length n are sent into two BiLSTMs, both with hidden sizes of h. The two BiLSTMs generate two sequence of $2h$-dimensional output vectors of length n and m respectively. Note that the reason for $2h$ dimension is that we concatenate the output of both directions together to aggregate the information of both directions.

2.3 Multi-Perspective Context Matching

The encoded sequence of words will be processed by the Multi-Perspective Context Matching (MPCM). This stage compares the embedding of the passage context with the question with multi-perspectives.

The first kind of context matching is the Parameterized Cosine defined by Equation (1). It computes the cosine similarity between two weighted vectors.

$$m_k = \cos(W_k \circ v_1, W_k \circ v_2) \tag{1}$$

Then the vectors are concatenated together after applying Pooling including max, mean etc. to form the resulting matching vector. The computation is defined in (2)

$$\vec{m}_j^{max} = \max_{i \in \{1,2,\ldots,m\}} f_m(\vec{h}_j^p, \vec{h}_i^p; W^3) \tag{2}$$

2.4 Aggregation and Prediction

The matching vector will be sent into another BiLSTM with the same hidden size as the first two BiLSTMs. The goal of this stage is to aggregate the information in the matching vector from multiple perspectives. The output of this BiLSTM will be sent into two different softmax classifiers. We reason we only use one BiLSTM for this stage is that intuitively, most the information needed to predicted the start and end position of the answer should be common, thus only the last softmax classifier needs to be different. Using two different BiLSTMs for predicting start and end position is worth studying and will be left as future work.

2.5 Prediction Optimization

After getting the prediction score, we optimized the prediction at test time by adjusting prediction result empirically. The first way we proposed is to average the predicted start position and end position based on their prediction scores. This approach intuitively denoise the output result. The second way we proposed is to simply set the end position to the maximum of predicted start and end position. This approach eliminate the situation where the end position accidentally resides before the start position. The third approach we proposed is to re-estimate the start and end position by

searching the neighborhood of predicted start and end position. This approach intuitively assumes the correct answer is more inclined to be short. As illustrated in Fig 2, when the distance between the predicted start and end position is too large or when the end position accidentally resides before the start position, we introduce the a heuristics of re-estimating the start and end position by comparing the value of the following three terms:

$$\max_{i \in \{0,1,...n\}} (P_{start}(end - i)) + P_{end}(end) \tag{3}$$

$$\max_{i \in \{0,1,...n\}} (P_{end}(start + i)) + P_{start}(start) \tag{4}$$

$$\alpha(P_{end}(end) + P_{start}(start)) \tag{5}$$

Here α is an empirical coefficient. When START ¿ END, α is set 0 because it is impossible. For N ¿ 15, α is set as 0.7. We did not try other values on the dev set.

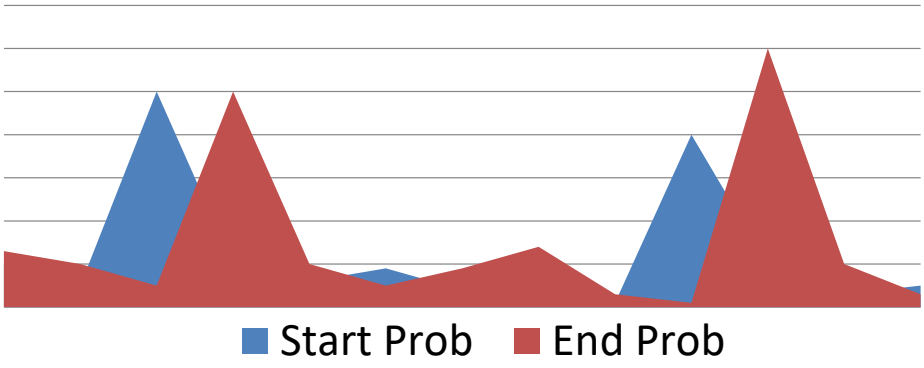

Figure 2: The example situation where re-estimation is necessary

3 Experimental Result

We developed our own model structure rather than using the provided code template. The whole architecture uses Tensorflow [1] as training framework. We introduced dropout to all BiLSTM used in the model. The whole model was trained for 4 epochs on one GTX 1080 GPU. The experimental results are listed in Table1.

Table 1: Experiment result

	Exact Match (EM)	**F1 Score**
Original paper's [4]	66.1	75.8
W\o optimization	47.52	58.13
Start,End = mean(Start, End)	47.2	58.27
End=max(Start,End)	48.22	59.5
Re-estimation	49.5	61.27

As we can see, before applying prediction optimization, we can achieve EM score of 47.52% and F1 score of 58.13% on development set. This result was obtained without using several tricks in the original paper.

Using the heuristics of averaging start position and end position separately will improve the F1 score a little, however, it will harm the EM score. One explanation for this phenomena is that denoising the score will help the prediction in general but the Exact Match still relies more on the most confident prediction.

Using the heuristics of averaging setting end position to the start position when it's predicted to be reside before start position can improve EM score by 0.7 and F1 score by about 1.4. This intuitively works since it eliminated the impossible predictions.

Using the heuristics of re-estimation describe in Section 2 can improve the EM score by 2.3 and F1 score by 3.0. This result shows the assumption that the correct answer is short and setting priority of searching in the neighborhood works well in improving prediction accuracy.

4 Observations and Discussions

4.1 The overfitting problem

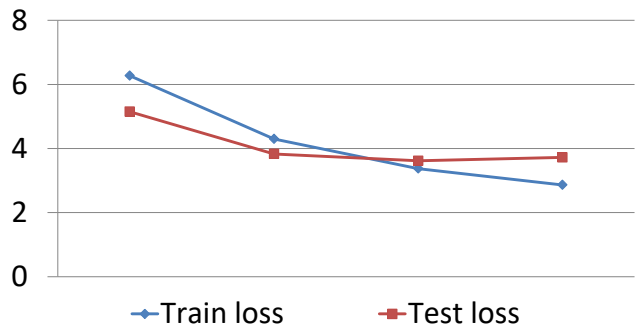

Figure 3: The curve of train & dev loss

The training loss curve is illustrated in Figure 3. As we can see, after only 3 epochs of training, the testing loss drops below the training loss, although we used dropout in all layers. We used several methods trying to deal with this problem but none of them is effective, including:

- Further increase the dropout ratio from 0.2 to 0.7. However the gap between train loss and dev loss do not seem to decrease.
- Set weight decay as 5e-4. The overfitting problem is substantially alleviated, but the dev accuracy remains the same or even a bit lower.

4.2 Predicting the length

Inspired by Faster R-CNN[3], we added one more feed-forward neural network to predict the log-length of the answer in a regression manner(use L2 loss). During the testing phase, the predicted log-length serves as a prior with Gaussian distribution. However we find F1 score is increased by a little(less than 1%) but Exact Match score was decreased by about 5%. It implies that length-based prediction is not accurate enough for this type of tasks.

Acknowledgement

The author would like to thank the teaching staffs of CS224N winter 2017 for contributing to this wonderful course.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

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

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[4] Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. Multi-perspective context matching for machine comprehension. *arXiv preprint arXiv:1612.04211*, 2016.