IRTA Practical Assignment: Deep Learning for Answer Selection

Hainan Yu, Zhenyu Guo, Yanfang Hou

20/05/2020

1 Problem and Dataset

Answer Selection (AS) is a task that enables selecting the best answer from a set of candidate answers for the given question. We apply different neural networks on the AS task and compare their performance.

We conduct the experiments based on the Microsoft Research WikiQA Corpus [11], which is a new publicly available set of question and answer pairs. It consists of 3047 questions and 29259 labeled sentences. Since there are only few negative answers for each question, we extend it to 50 negative answers by randomly sampling from the entire answer pool.

2 Data preprocessing

The data is divided into train, dev and test set which consist of 20360, 2733 and 6166 question-answer pairs respectively. We exclude the questions with no correct answers and truncate each sentence to 40 tokens.

3 Method

In this section we briefly describe the models we use: Bidirectional Long Short-Term Memory (BiLSTM), Convolutional Neural Network (CNN), Simple Recurrent Neural Network (SimpleRNN) and Gated Recurrent Unit (GRU).

3.1 BiLSTM

BiLSTM is an extension of Long Short-Term Memory (LSTM) [4]. Therefore we first describe LSTM method briefly.

3.1.1 LSTM

The original LSTM is a variant of RNN to overcome the error back-flow problems (gradient vanish) of RNN [4]. Our implementation of LSTM is similar to Tan et al. [9] and Graves et al. [3].

Given an input sequence $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_1, ..., \mathbf{x}_n\}$ with a length n (In our case, each $\mathbf{x}_t$ is a d dimension word d dimension embedding vector). The next formulas show how LSTM returns the hidden vectors $\mathbf{h}_t \in R^{H\times d}$ with a size H:

$$\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i)\\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o)\\
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f)\\
\widetilde{C_t} &= tanh(W_c x_t + U_c h_{t-1} + b_c)\\
C_t &= i_t \odot \widetilde{C_t} + f_t \odot C_{t-1}\\
h_t &= o_t \odot tanh(C_t)
\end{aligned} \tag{1}$$

where input i, forget f, and output o are three gates of LSTM networks and σ is *sigma* activation function, and $W \in R^{H\times d}$, $b \in R^{H\times 1}$, $U \in R^{H\times H}$ are all networks parameters, and C_t is the cell state, and $\odot$ denotes the element-wise multiplication. The current input information will be selected by three gates and conveyed to next time step.

3.1.2 BiLSTM: Enhancement for LSTM

Since unidirectional LSTM has the weakness that current hidden vector does not contain the contextual information of the future step, BiLSTM is invented as a remedy. BiLSTM utilizes both of the previous and future context by processing the sequence on two directions and generate two independent sequences of hidden vectors. The hidden vector at each time step is the concatenation of the hidden vectors from two direction, i.e. $h_t = \overrightarrow{h_t}||\overleftarrow{h_t}$. The output of BiLSTM is a sequence of hidden vectors $H \in R^{L\times 2H}$ where L is the maximum of the word sequence of our input sentences [2, 9, 10].

Figure 1 [2] shows the model structure that we use in answer selection task. The first layer of network is to transform two sequences of words for question and answer into fixed-size word embeddings. In our experiment, we use the pre-trained glove model [8] to represent each word with 300-dimensional vector. Then we fed the two sequences of word embeddings into BiLSTM layer and obtain two hidden vectors. Next we apply a max pooling layer to obtain the final vector representations for question and answer. Finally we use the outputs o_q and o_a to calculate their cosine similarity $cos(o_q, o_a)$.

The loss function we use is the triplet hinge loss, shown in Equation 2 according to Tan et al. [9] and Tran and Niederée [10].

$$L = max\{0, M - cos(q, a_+) + cos(q, a_-)\} \tag{2}$$

where M is the margin and *cos* means the cosine similarity between two input vectors.

We set this loss score as our customized loss to update our model to get a better representation of the input sentences, which can help our model shorten the distance between the question and positive answer and increase the distance between the question and negative answer.

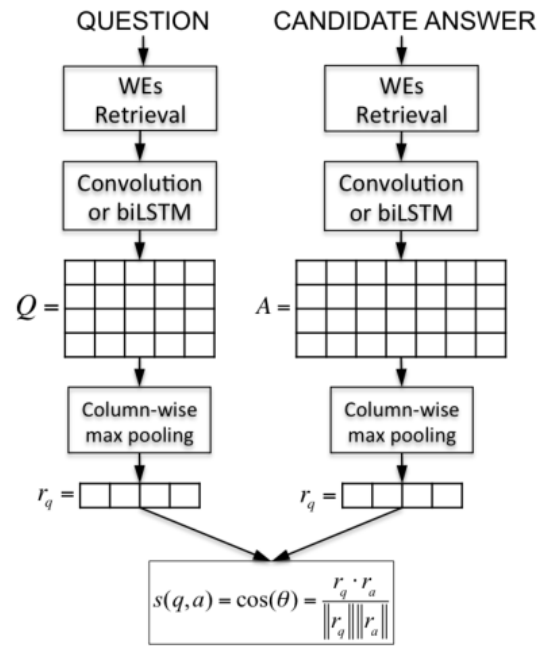

Figure 1: LSTM or CNN based model structure

3.2 CNN

Convolutional neural networks are a type of neural networks that are widely used in processing images. 1D CNN can be applied for Natural Language Processing although it is probably not as promising as RNN and LSTM. In this project, we apply it as a basic method to compare with biLSTM layer. A convolutional neural network consists of an input layer, convolutional layers and an output layer. The output layer is given by [7]:

$$n_{out} = \left[\frac{n_{in} + 2p - k}{s}\right] + 1 \tag{3}$$

where n_{in} and n_{out} represents number of input features and output features, k is the kernel size, p is the padding size and s is the stride size. Comparing to biLSTM and RNN, CNN requires fixed length input and the last state can not be the input for the next step.

3.3 Simple RNN

We replace the biLSTM layer with simple RNN layer as comparison. Recurrent neural networks are a class of neural networks that deal with sequential information. It allows previous states to be used as inputs to the next step. For each timestamp, the hidden state is given by

$$h_t = f(h_{t-1} \cdot W_{rec} + X_t \cdot W_x) \tag{4}$$

where h_t is the state at time t, X_t is an input at time t, W_{rec} and W_x are shared coefficient and f is activation function [6]. Hence, h_t captures information about what happened in all the previous time steps. Compared to biLSTM, simple RNN is not capable of capturing past information from a long time ago in practice. Due to single direction, it is also not able to capture future information.

3.4 GRU

We try GRU instead of biLSTM to build the model and observe its performance. Gated Recurrent Units (GRU) are pretty similar to LSTM, which are also used to solve long-term dependency problem. Compared to LSTM, GRU gets rid of the cell state and only have two gates, update gate and reset gate. Given the previous state h_{t-1} and x_t, the next hidden state h_t is generated as follows [6]:

$$\begin{aligned} z_t &= \sigma(W_z x_t + U_z h_{t-1}) \\ r_t &= \sigma(W_r x_t + U_r h_{t-1}) \\ \widetilde{h}_t &= tanh(r_t \odot U h_{t-1} + W x_t) \\ h_t &= (1 - z_t) \odot \widetilde{h}_t + z_t \odot h_{t-1} \end{aligned} \tag{5}$$

where a new memory $\widetilde{h}_t$ is the combination of a new input with the previous memory, r_t is the reset gate that determines how important h_{t-1} is to the summarization $\widetilde{h}_t$, z_t is the update gate that decides how much previous memory should be carried forward to the next state, and h_t is finally generated by using the previous hidden state h_{t-1} and the new memory $\widetilde{h}_t$.

4 Experiment

4.1 Baselines

We use tf-idf method on the entire dataset as our baseline. We represent each sentence as a vector by using tf-idf to weight each term in the sentence.

The tf-idf weighting of each term in the sentence is given as follows [5].

$$\begin{aligned} tf_{t,s} &= log(1 + count(t, s)) \\ idf_t &= log(N/df_t) \\ tf - idf_{t,s} &= tf_{t,s} \times idf_t \end{aligned} \tag{6}$$

where $count(t, d)$ refers to the occurrence frequency of the term t in the sentence s, N is the amount of sentences and df_t represents the total number of sentences that contain the term t. Then we calculate cosine similarity between the question and candidate answers.

In our experiment, we first vectorize the documents and achieve the tf-idf transformation. The obtained matrix has the size of $Ns \times Nt$, where Ns represents the amount of sentences and Nt refers to the total number of terms that occur in the dataset. Because of the sparsity of the matrix, we apply the $SparseMatrixSimilarity$ function to calculate the similarity matrix between different sentences.

4.2 Metrics

We evaluate the model performance by using the Precision@1, Mean reciprocal rank (MRR) and Mean Average Precision (MAP). Specifically, for each test question, there are 50 candidate answers consisting of the correct answers and randomly chosen negative answers. we sort the candidate answers in descending order based on the cosine similarity of each question-answer pair. Then we use the following metrics to measure the performance.

- Precision@1: This metric is determined based on whether the top predicted answer is ground truth.
- MRR: This metric is given by $MRR = \frac{1}{|Q|} \sum_{i=1} |Q| \frac{1}{rank_i}$, where $rank_i$ refers to the position of the first correct answer for the i-th question.
- MAP: For each test question, the average precision is the mean of the precision scores after each correct answer is retrieved. Then the MAP is the mean of the average precision scores for all test questions.

4.3 Result

Table 1: Parameter setting of CNN model

	filters	kernel size	dropout	L2	learning rate	batch size	epoch	margin
CNN	150	3	0.2	0.005	0.0005	64	10	0.2

As shown in table 3, the tf-idf vector model performs pretty well and even better than CNN model. We guess the reason is that it can efficiently exclude

Table 2: Parameter settings of different RNN models

	hidden units	dropout	L2	learning rate	batch size	epoch	margin
simple RNN	100	0.5	0.005	0.00025	64	15	0.2
GRU	100	0.5	0.005	0.00025	64	11	0.2
BiLSTM	141	0.5	0.005	0.001	128	2	0.2

Table 3: Experimental results on WikiQA dataset

	Validation			Test		
	MAP	MRR	Precision@1	MAP	MRR	Precision@1
TF-IDF	0.5048	0.5113	0.3254	0.5220	0.5399	0.3817
CNN	0.4895	0.4895	0.3429	0.4645	0.4645	0.3265
simple RNN	0.5650	0.5706	0.4206	0.5479	0.5629	0.4025
GRU	0.6217	0.6331	0.4603	0.6169	0.6380	0.4896
BiLSTM	0.6267	0.6355	0.4921	0.6159	0.6271	0.4564

the candidate answers that do not have the same words with the question. Hence it can filter some answers that are significantly different from the question, but might be difficult to distinguish sentences with much lexical similarity.

For the performance of neural networks, the three out of them (simple RNN, GRU, and BiLSTM) outperform the baseline TF-IDF model on both the validation and test data sets, which attributes to the fine construction and learning ability of the neural networks. We find that the vanilla CNN model performs worse than the baseline. Considering that the pure CNN model is better at processing data that has close spatial connections, while RNNs are well suited to encode sequential information and long-range context dependency [12]. Our data set is a kind of question-answering data set and is characterized by temporal connections, therefore RNNs are able to play a role on it.

We can also observe that our GRU model and BiLSTM models have comparable results on both validation and test data sets. These two models have the similar architecture with some differences in design. They have two advantages compared with traditional RNN. First, Both of them can easily remember the existence of a specific feature in the input stream for a long series of steps and control how much the units update its activation or content. Second, they can back-propagate the error easily therefore avoiding gradient vanishing [1]. The overall results on test sets show GRU performs better slightly than LSTM, which could because of the small scale of our data set.

In terms of the configuration of hyperparameters, simple RNN and GRU have similar settings, which contains smaller sizes of hidden units, learning rate, batch size, and a longer training procedure (bigger epoch sizes) than BiLSTM. The model CNN has its own hyperparameter design.

5 Conclusion

In this report, we study to tackle the answer selection problem by applying different approaches. The experimental results show that BiLSTM and GRU outperform other neural networks on WikiQA dataset, since they can efficiently process the sequential data and extract long-term dependency. By contrast, CNN performs worst as it is not able to capture sequential information. For future work, we want to combine the CNN with LSTM network on answer selection task. In addition, we would like to deploy the attention mechanism to improve the performance of the model.

References

[1] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *ArXiv*, abs/1412.3555, 2014.

[2] Cícero Nogueira dos Santos, Ming Tan, Bing Xiang, and Bowen Zhou. Attentive pooling networks. *ArXiv*, abs/1602.03609, 2016.

[3] Alex Graves, Abdel rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[5] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval.* Cambridge university press, 2008.

[6] Richard Socher Lisa Wang Amita Kamath Milad Mohammadi, Rohit Mundra. Language models, rnn, gru and lstm, 2019.

[7] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *ArXiv e-prints*, 11 2015.

[8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[9] Ming Tan, Bing Xiang, and Bowen Zhou. Lstm-based deep learning models for non-factoid answer selection. *ArXiv*, abs/1511.04108, 2015.

[10] Nam Khanh Tran and Claudia Niederée. Multihop attention networks for question answer matching. *The 41st International ACM SIGIR Conference on Research Development in Information Retrieval*, 2018.

[11] Yi Yang, Wen-tau Yih, and Christopher Meek. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018, 2015.

[12] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. *ArXiv*, abs/1702.01923, 2017.