A Question Answering System on SQuAD Dataset Using End-to-end Neural Network

CS297 Report

Student: Bo Li

Advisor: Dr. Chris Pollett

Date: Dec 2017

TABLE OF CONTENTS

1 Introduction

Question Answering(QA) is about making a computer program that could answer questions in natural language automatically. QA techniques are widely used among search engines, personal assistant applications on smart phones, voice control systems and a lot more other applications. In recent years, more end-to-end neural network architectures are built to do question answering tasks. In contrast, traditional solutions use syntactic and semantic analyses and hand made features. End-to-end neural network approach gives more accurate result. However, traditional ways are more explainable. The Stanford Question Answering Dataset (SQuAD) is used in this project. It includes questions asked by human beings on Wikipedia articles. The answer to each question is a segment of the corresponding Wikipedia article[1]. In total, SQuAD contains 100,000+ question-answer pairs on 500+ articles[1]. The goal of this project is to build a QA system on SQuAD using an existing end-to-end neural network architecture. If there is still time left after finishing the QA system, I will review related literatures and try to come up with an improved architecture.

This report is about my progress in CS297. Section 2 corresponds to deliverable 1, which is a study on some very basic mathematics of neural network. Section 3 corresponds to Deliverable 2, in which I did word embedding of Chinese classic poems. Please be noted, the project topic was changed in November 2017 and Section 3 was not aimed at the current topic. Section 4 and Section 5 correspond to Deliverable 3 and 4, which are about the current topic. Section 6 includes my comment on my work in CS297 and my plan for next step.

2 Calculation of Back Propagation

The purpose of this deliverable is to understand the mathematic foundations of neural network, which is the technical approach of this project. I fulfilled this purpose by doing back propagation on a dummy feed forward neural network example below.

$$\hat{y} = softmax(z_2)$$

$$z_2 = h \cdot W_2 + b_2$$

$$h = sigmoid(z_1)$$

$$z_1 = x \cdot W_1 + b_1$$

Define loss

$$J(W_1, b_1, W_2, b_2, x, y)$$

$$= cross\_entropy(y, \hat{y})$$

$$= -\frac{1}{D_y} \sum_{i=1}^{D_y} y_i \times \log \hat{y}_i$$

After using chain rules multiple times, I got

$$\frac{dJ}{dz_2} = \hat{y} - y$$

$$\frac{dJ}{db_2} = \frac{dJ}{dz_2}$$

$$\frac{dJ}{dh} = \frac{dJ}{dz_2} \cdot W_2^T$$

$$\frac{dJ}{dW_2} = h^T \cdot \frac{dJ}{dz_2}$$

3 Implementation of Word Embedding

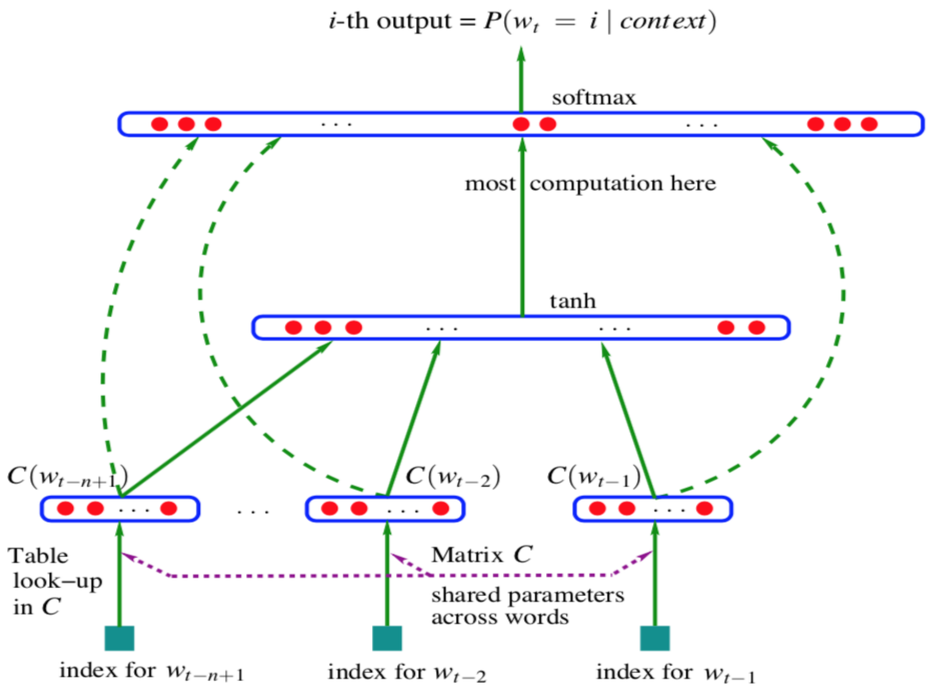

Figure 1: Neural architecture: $f(i, w_{t-1}, ..., w_{t-n+1}) = g(i, C(w_{t-1}), ..., C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector[2]

Word embedding is an important part of applying neural network to natural language processing. Although this deliverable is done for the old topic, which is replaced by the current topic in November 2017, understanding word embedding is also an essential part of the current topic.

Word embedding is a way to map each word to a feature vector in a continuous space. The dimension of the continuous space is much lower than the dimension of one-hot vector, which is comparable to the vocabulary size. Also, the distance between two word feature vectors could tell how likely the two corresponding words appear in same context.

Word embedding is originally introduced by Bengio et al in [2]. They propose a neural probabilistic language model(NPLM) which is illustrated in Fig.1. The training set is a sequence of words $w_1, ..., w_T$ where $w_t \in V$ and V is the vocabulary. The purpose is to train a model f such that $\hat{P}(w_t|w_{t-1}, ..., w_{t-n+1}) = f(w_t, ..., w_{t-n+1})$. The computation of $f(w_t, ..., w_{t-n+1})$ is divided into two parts. First, we map each w to a distributed feature vector by selecting the corresponding row in C.

$$x = (C(w_{t-1}), ..., C(w_{t-n+1}))$$

Second, we maps x to $f(w_t, ..., w_{t-n+1})$.

$$y = b + W \cdot x + U \cdot tanh(d + H \cdot x)$$

$$f(w_t, ..., w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

The loss function to minimize is

$$L = -\frac{1}{T} \sum_t \log f(w_t, ..., w_{t-n+1})$$

At the present time, an simplified architecture proposed by Mikolov et al in [3] is widely used. The main difference between it and NPLM is Skip-gram removes tanh layer.

I implemented in Python both the NPLM model without Noise Contrastive Estimation (NCE) loss and skip-gram model with NCE loss. I use a collection of 284899 classic Chinese poems as the corpus.

Here are some information about the skip-gram together with negative sampling implementation. Training each epoch costs about 8 minutes. After about 5 epochs, the valid loss reaches the lowest. I selected 200 most common words and calculated cosine similarity between each two words pair to get 40000 word pair cosine similarity. Table 1 lists top results among the 40000 results . According to my knowledge of Chinese classic poems, in most word pairs in Table 1, the two characters have high probability to appear in same context. As such, I think the model is implemented correctly.

作后0.999374	当少0.999315	同好0.999307	闻好0.999266
同少0.999261	愁闲0.999212	好少0.999189	红叶0.999121
复少0.999101	当复0.999071	故少0.999031	醉闲0.999025
同闻0.999023	出开0.999002	空入0.999001	起发0.99899
平小0.998955	亦应0.998954	雪叶0.998952	竹叶0.998946
小龙0.998945	发晚0.998937	分歌0.99893	起晚0.998928
寒满0.998914	过向0.998909	当真0.998894	入阴0.99889
愁后0.998882	情言0.998881	尽到0.998878	当故0.998865
到起0.998859	闻少0.998846	旧少0.998841	当犹0.998839
开阴0.998839	复物0.998835	亦还0.998833	言以0.998825

Table 1: Highest Similarities Between 200 Most Common Words

4 Understanding Online Evaluation Environment and Setting Up Developing Environment

4.1 Understanding the Online Evaluation Environment

To evaluate a model, a prediction Python script should be submitted through Codelab.

As such, training and prediction must be separated. After training the mode, a tensorflow graph should be saved to disk. Then the prediction script should restore the tensorflow graph to make prediction on test data. This requires concise names and scopes for important nodes in the graph.

4.2 Docker

There are two concerns about development infrastructure. First, ascending complexity and software version dependencies might cause problem. Second, I might need to use cloud GPU to train the model in the future.

Docker helps solving this problem By using Docker, I can list all software dependencies in Docker file, build a Docker image using the Docker file, and then run a Docker container using the Docker image. Docker also supports potability between different devices.

Below is the content of the Docker file I used.

```
FROM tensorflow/tensorflow
RUN pip install joblib
RUN pip install nltk
RUN pip install tqdm
RUN pip install pyprind
RUN python -m nltk.downloader --dir=/usr/local/share/nltk_data perluniprops punkt

WORKDIR /297And8QuestionAnswer
```

5 Question Answering System Architecture

5.1 Review of Paper [4]

Wang and Jiang [4] proposes an end-to-end neural network model on SQuAD dataset. While predicting, the inputs to the model are test data and pretrained word embeddings, the outputs are the predicted answers. While training, the inputs are train data and word embeddings, the outputs are losses to optimise. GloVe is used to do word embedding. The word embedding is not updated during training.

Model architecture includes three layers-the LSTM preprocessing layer, the match-LSTM layer and the Answer Pointer(Ans-Ptr) layer.

The LSTM preprossing layer encode each word sequence in passage and question to a sequence of hidden states using a standard one direction LSTM. The passage and question are processed separately.

$$H^p = \overrightarrow{LSTM}(P)$$
$$H^q = \overrightarrow{LSTM}(Q)$$

where

$$P \in R^{d \times p} : passage$$
$$Q \in R^{d \times q} : question$$
$$H^p \in R^{l \times p} : encoded\ passage$$
$$H^q \in R^{l \times q} : encoded\ question$$
$$p : length\ of\ passage$$
$$q : length\ of\ question$$
$$l : dimension\ of\ LSTM\ hidden\ states$$
$$d : dimension\ of\ word\ embedding$$

The match-LSTM layer uses the model in paper [5]. In this layer, a word-by-word attention mechanism and a LSTM are used together to encode hidden presentations of both passage and question to one sequence of hidden states that indicate the degree of matching between each token in the passage and each token in the question. To be specific,

$$\overrightarrow{G} = tahn(W^q H^q + (W^p h_i{}^p + W^r \overrightarrow{h_{i-1}{}^r} + b^p) \otimes e_q)$$
$$\overrightarrow{\alpha_i} = softmax(w^t \overrightarrow{G_i} + b \otimes e_q)$$

where

$$W^q, W^p, W^r \in R^{l \times l}$$
$$b_p, w \in R^l$$
$$b \in R$$
$$\overrightarrow{h_{i-1}{}^r} \in R^l : one\ column\ of\ H_p$$

and

$$\overrightarrow{z_i} = \begin{bmatrix} h_i{}^p \\ H^q \overrightarrow{\alpha_i}^T \end{bmatrix} \in R^{2l}$$
$$\overrightarrow{h_i{}^r} = \overrightarrow{LSTM}(\overrightarrow{z_i}, \overrightarrow{h_{i-1}{}^r})$$

After iterating between getting attention vector $\overrightarrow{\alpha_i}$ and getting hidden state $h_1{}^i$ p times, we get $[h_1{}^r, ..., h_p{}^r]$. Concatenate them to get

$$\overrightarrow{H_r} = [h_1{}^r, ..., h_p{}^r] \in R^{l \times p}$$

.

To go over passage from both directions to get context information both before and after each word, go over H_p from right to left to get $\overleftarrow{H_r}$. Then concatenate $\overrightarrow{H_r}$ and $\overleftarrow{H_r}$ to get

$$H_r = \begin{bmatrix} \overrightarrow{H_r} \\ \overleftarrow{H_r} \end{bmatrix} \in R^{2l \times p}$$

The Answer Pointer layer is motivated by the Pointer Net in paper [6]. It has a similar structure with match-LSTM. However, instead of aiming at a sequence of hidden states, Ans-Ptr layer aims at the weight vector. Here I only explain the boundary model, which I will implement.

$$F_k = tahn(VH_r + (W^a {h_{k-1}}^a + b^a) \otimes e_p)$$
$$\overrightarrow{\beta_k} = softmax(v^t F_k + c \otimes e_p)$$

where

$$V \in R^{l \times 2l}$$
$$W^a \in R^{l \times l}$$
$$b_a, v \in R^l$$
$$c \in R$$
$$\overrightarrow{{h_{k-1}}^a} \in R^l : hidden\ state\ at\ positiom\ i\ of\ answer\ LSTM$$

and answer LSTM is

$$\overrightarrow{{h_k}^a} = \overrightarrow{LSTM}(H^r \beta_k^T, h_{k-1}^a)$$

By iterating between the attention mechanism and the answer LSTM two times, we could get β_0 and β_1. Let a_s denote start index of the answer, and a_e denote the end index, then we have

$$p(a|H^r) = p(a_s|H_r)p(a_r|H_r) = \beta_{0,a_s} \times \beta_{1,a_e}$$

where

$$\beta_{k,j} = jth\ token\ of\ \beta_k$$

To train the model, the loss function

$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log p(a^n|H^r)$$

is minimized.

5.2 Implementation Architecture

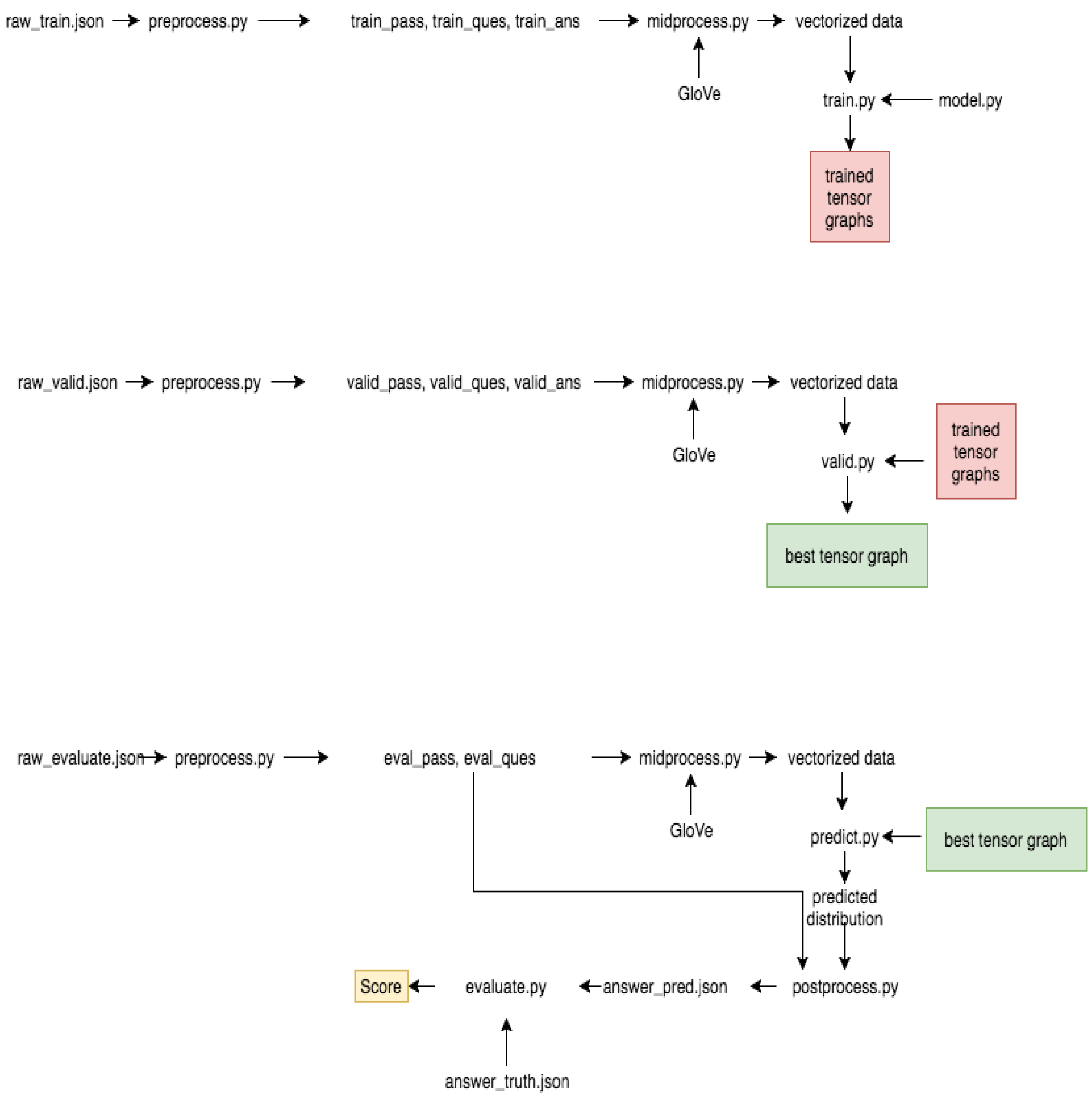

Figure 2: Implementation Architecture of Paper [4]

As indicated in Fig.2, the training pipeline, validation pipeline and evaluation pipeline are separated. This not only separates development and deployment, but also make the large system easier to debug.

Take the training pipeline as example. The `preprocess.py` reads a json file, splits out passages, questions and answers, tokenizes them, and outputs each passage, question or data in token sequence. The `midprocess.py` reads token sequences and word embedding GloVe, vectorized token sequences to vector sequences, trims or pads the sequences, makes mask tensor, and outputs feed-ready data. Here, GloVe word vectors are pre-trained word vectors using GloVe algorithm [7]. GloVe algorithm is an unsupervised learning algorithm to train word vectors. In addition to a neural network, it also uses co-occurrence statistics from a corpus. The `train.py` takes feed-ready data and `model.py`, iterate through different number of epochs, learning rates, training optimizer and so on, and store trained models to disk.

Since Tensorflow graphs are shared for the training, validation and evaluation steps, the feed data in the three steps must be consistent with the graphs. Since a graph is defined using `pass_max_length`, `batch_size`, `embed_size`, and `num_units`, which is called l in theoretical model, train, valid and evaluate data are vectorized, padded, and divided to mini batches in same file `midprocess.py` using same `pass_max_length`, all data must be divided to same `batch_size`. The `num_unit` does not relate to shape of feed data.

To achieve batched training, paragraphs should be padded to a same length. Similarly, questions are also padded the a same length. As such, the model in implementation has some difference with the theoretical model explained in 5.1.

In preprocessing layer,

$$H^p = H^p \circ passage\_mask$$

$$H^q = H^q \circ question\_mask$$

In match-LSTM layer,

$$\overrightarrow{\alpha_i} = softmax((w^t \overrightarrow{G_i} + b \otimes e_q) \circ question\_mask)$$

$$H_r = H_r \circ passage\_mask$$

In Ans-Ptr layer,

$$\overrightarrow{\beta_k} = softmax((v^t F_k + c \otimes e_p) \circ passage\_mask)$$

6 Summary

I have accomplished several milestones in CS297. First, I have implemented Skip-gram model on Chinese classic poems data successfully. Second, I have reviewed paper [4] and mastered feed forward network, RNN, LSTM, and attention mechanism. Third, I have designed a nice architecture and wrote some coding for the Question Answering system.

What I have done in this semester builds a good foundation for implementing a Question Answering system. Using the architecture I designed in 5.2, I could code up the Question Answering system. This is what I will do in CS298.

References

[1] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[2] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[4] S. Wang and J. Jiang, "Machine comprehension using match-lstm and answer pointer," *arXiv preprint arXiv:1608.07905*, 2016.

[5] ——, "Learning natural language inference with lstm," *arXiv preprint arXiv:1512.08849*, 2015.

[6] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2692–2700.

[7] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.