COMP9417 - Assignment 2

Avinash K. Gupta, Yuyang Shu, Maria Oei

June 4, 2017

Contents

1 Introduction

With over 100 millions of monthly visitor in Quora, it is inevitable that many people are asking similar questions. This has become an issue as user has to read through responses to many questions in order to find the best answer. The aim of this project is to implement algorithms to identify 2 similar questions which can help Quora in improving user experience by finding high quality answers to questions.

The 2 approaches used in this project were perceptron learning and LSTM. In the perceptron learning, we used 3 inputs which measured semantic similarity, word order and word overlaps between 2 sentences. In LSTM, we provide word-embedding vectors supplemented with synonymic information and calculates the manhattan distance between the vectors.

We will see that for our case the performance of the 2 models were quite similar. However, there are other research done on LSTM where it has been shown that LSTM has the potential to performs well with enough training and with careful selection of initial weight.

2 Methodology

In this project, we used WordNet as the main lexical database. Using WordNet, we were able to extract the part-of-speech tags for a word as well as their synset (synonym set). This played an important role in determining the similarity of 2 words as we would see later.

2.1 Data Preparation

The training data provided contained pairs of questions and indicators to determine if the two questions are similar. In the first model, the data was first tokenized into list of words and stemmed to their base form. As a word could be associated to multiple synset, we used Lesk algorithm to choose the correct synset given the context of the sentence. In Lesk algorithm, for each possible synset we collected its hypernyms and hyponyms into a list. We then counted the number of common words from this list with the remaining words in the sentence. The synset with the highest number of overlap will then be chosen to be the best synset for the word. Once this was done, stop words were removed and the tokens were sent to the function which computed the semantic and word order score of the 2 sentences. Please see appendix for more details on how to calculate these measures.

2.2 Training

To train the data we used the measures from the above procedure as the features. The original model proposed in [] was designed to map each feature to be between 0 and 1. There was also a constraint put on the weights to ensure that the sum of the weights was 1. However, we tried to applying perceptron learning in order to automate the learning process. The perceptron initialized the weights to be small random variable with 0.1 learning rate and 1000 iteration.

3 Result & Discussion

3.1 Conclusion

3.2 Limitation & Improvement

4 References

Jonas Mueller, Aditya Thyagarajan (2016). Siamese Recurrent Architectures for Learning Sentence Similarity. AAAI Conference on Artificial Intelligence, pp.2786-2792. Available at: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12195 .

Li, Y., McLean, D., Bandar, Z., O'Shea, J. and Crockett, K. (2006). Sentence similarity based on semantic nets and corpus statistics. IEEE Transactions on Knowledge and Data Engineering, [online] 18(8), pp.1138-1150. Available at: http://ieeexplore.ieee.org/document/1644735/.

GitHub. (2017). aditya1503/Siamese-LSTM. [online] Available at: https://github.com/aditya1503/Si
LSTM [Accessed 4 Jun. 2017].

5 Appendix

5.1 Method 1 : Perceptron

The first model was a a simple perceptron with 2 inputs which were based on [insert reference to paper here]. The inputs chosen measured the semantic similarity and the word order information of the 2 sentences. For each sentences pair in the dataset, both semantic similarity and word order score were calculated. The perceptron weights were then trained based on these measures.

5.1.1 Words Similarity

A **path length** between 2 words is the number of synsets we visit from one word to another. For example, in the figure below, to get from boy to girl we have to visit boy - male - person - female - girl. Therefore, the path length of 'boy' and 'girl' is 4. 'Person' is called the subsumer of 'boy' and 'girl'. If there are more than 1 path, we will consider the shortest path and the corresponding subsumer is called the **lowest subsumer**.

Let l denote the shortest path and h denote the depth of the lowest subsumer. The similarity between 2 words w_1 and w_2 is therefore measured by

$$s(w_1, w_2) = f_1(l)f_2(h)$$

where $\alpha, \beta \in [0, 1]$ and

$$f_1(l) = e^{-\alpha l}$$
$$f_2(h) = \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}}$$

5.1.2 Semantic Similarity

Let T_1 and T_2 be the 2 sentences and T is a set of distinct words in T1 and T2. For each sentence, we will calculate the vector s_k which the same length as T. For each word w_i in

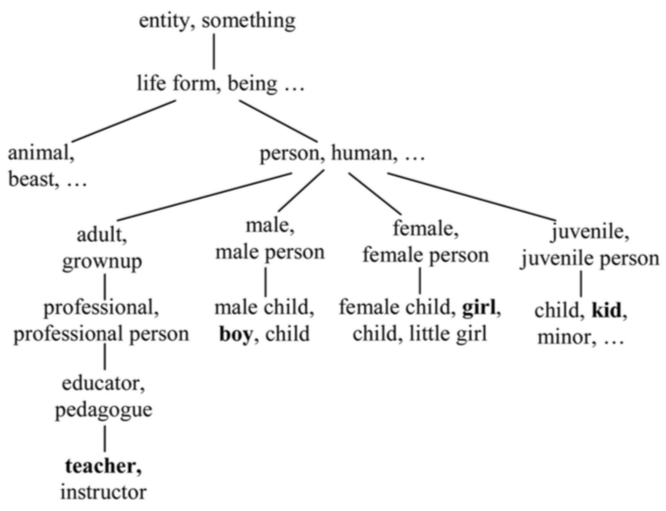

Figure 1: Synset Tree

T, we assign 1 to the corresponding element in s_k if w_i is in T_k and μ_i otherwise. μ_i is the similarity score between w_i and the most similar word in T_k calculated based on the similarity score above. Once s_1 and s_2 are calculated, the overall semantic similarity score is calculated by

$$S_s = \frac{s_1.s_2}{||s_1||.||s2||}$$

5.1.3 Word Order

Similar to the semantic similarity, we calculate the vectors r_1 and r_2 for each of the sentences. For each word w_i in T, we set the i^{th} element in r_k to equal to the position of w_i in T_k. If w_i is not in T_k then we find the most similar word in T_k and assign the position of that word instead.

For both word order and semantic similarity measure, we define a threshold for the case where w_i is not in T_k. If the similarity score between w_i and the most similar word is less than the threshold, 0 will be assigned instead.

For the word order, the overall score is calculated by

$$S_r = 1 - \frac{||r_1 - r_2||}{||r_1 + r_2||}$$

5.2 Method 2 : LSTM - RNN

5.2.1 Overview

In this approach, we present siamese adaption of Long Short-Term Memory(LSTM) network for labeled data contains pairs of variable-length sequences. This model is used to get the semantic similairty between the sentences using complex neural network. For this applications, we provide word-embedding vectors supplemented with synonymic information to the LSTMs, which use a fixed size vector to convert the syntactic meaning of the sentence.

5.2.2 Model

It's a supervised learning model, where each data consists of pair of sequences $(x_1^{(a)}, ..., x_{n_a}^{(a)})$, $(x_1^{(b)}, ..., x_{n_b}^{(b)})$ of fixed size vectors along with a single label y(human labeled) for the pair. Note that sequences may be differenent length. These data will be pass to the model with a purpose of learning the semantics. In the above diagram, there are two networks $LSTM_a$ and $LSTM_b$, each of them process a sentence in a given pair and predict whether $LSTM_a = LSTM_b$ based on the similarity between the vectors. In each LSTM, the word vector is employ to the hidden layer and calculation is done and output is passed to the next hidden layer to remember the previous context and so on. In above figure, the hidden layer which is core part of our neural network. It performs operation listed below on each word vector.

- The first sigmoid function is used to decide which information to keep and what to ignore

$$f_t = sigmoid(W_i x_t + U_i h_{t-1} + b_i)$$

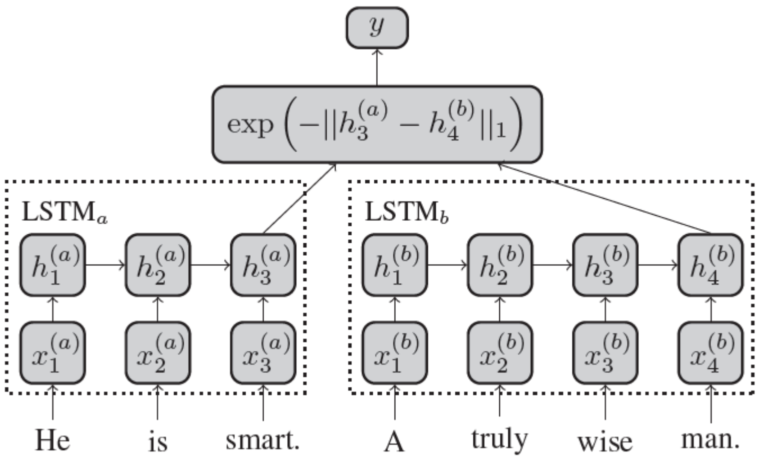
y
$\exp\left(-\|h_3^{(a)} - h_4^{(b)}\|_1\right)$
LSTM$_a$
LSTM$_b$
$h_1^{(a)}$ $h_2^{(a)}$ $h_3^{(a)}$
$x_1^{(a)}$ $x_2^{(a)}$ $x_3^{(a)}$
$h_1^{(b)}$ $h_2^{(b)}$ $h_3^{(b)}$ $h_4^{(b)}$
$x_1^{(b)}$ $x_2^{(b)}$ $x_3^{(b)}$ $x_4^{(b)}$
He
is
smart.
A
truly
wise
man.

Figure 2: lstm architecture

- Now it's time to update the old cell state into new cell state. The last step already decided what to do, in this step we just actually need to do it.

$$i_t = sigmoid(W_f x_t + U_f h_{t-1} + b_f)$$

$$\grave{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

- In this step, we will multiply old state c_{t-1} with f_t to forget things we decided to forget earlier. Then we add to new candidate values

$$c_t = i_t \odot \grave{c}_t + f_t \odot c_{t-1}$$

- Final step defines what we actually need to output based on out filtered cell state and then tanh(used to push values between -1 and 1) and multiply with sigmoid function to decide the parts we need to output.

$$o_t = sigmoid(W_o x_t + U_o h_{t-1} + b_o)$$

$$h_t = o_t \odot \tanh(c_t)$$

Above process are carried by both the LSTM and emploed to the Similarity Function given below, which calculates the Manhattan differnece between the ouputs.

$$g(h_{T_a}^{(a)}, h_{T_b}^{(b)}) = \exp(- \parallel h_{T_a}^{(a)} - h_{T_b}^{(b)} \parallel_1) \in [0, 1].$$

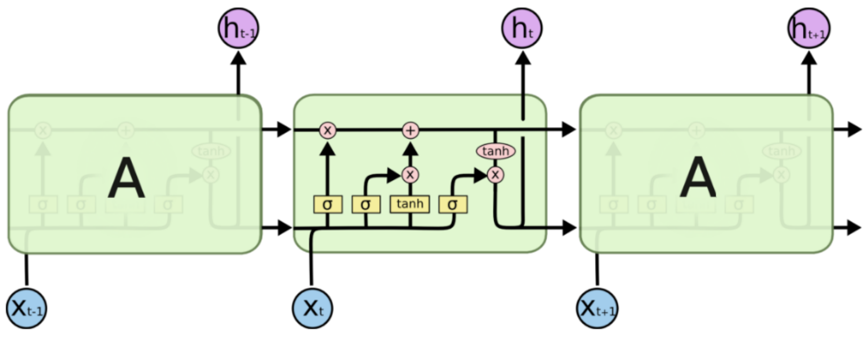

Figure 3: repeating module of hidden layer

5.3 Dependencies

1. Python packages

   - nltk
   - numpy
   - Scipy
   - gensim
   - Theano
   - cython

2. Tested System Configuration

   - intel i5 6200u
   - Intel HD Graphics 520
   - 8 GB Ram
   - 128GB SSD