NLA PROJECT - REPORT

Authors - Eesha Dutta and Paryul Jain
Date of Submission : 1st May 2019
20171104 and 20171083

1 Abstract

We applied deep learning framework to tackle the tasks of finding duplicate questions. We implemented some models following the siamese architecture using the popular recurrent network such as Long-Short Term Memory (LSTM), Bi-directional Long-Short Term Memory (BiLSTM) and Grated Recurrent Unit (GRU) to find the semantic similarity between questions. We started with a basic model and further extended the basic model into other models. We tried out different embeddings like Word2Vec , Glove etc and different similarity measures like Manhattan Distance, Softmax and Cosine distance on these models. We applied attention mechanism for getting better contextual meaning of the questions in one of the models. As neural models are data driven, we trained our models extensively by making pairs, such as question-question over a large-scale real-life dataset. We used a dataset consisting of 400K labeled question pairs which are published by a well known question-answer forum Quora. We evaluate our models based on metrics like accuracy, precision, recall, F1 scores. Our methods report significant similarities to the baselines and their improvements.
Click to see our **Code** and **Models**. Here's the **Github** Repository.

2 Introduction

Question-Answer forums are gaining more and more popularity day by day. Each forum itself becoming a community where people are more active than ever. As a result a large number of questions are posted each day, among them there are some questions which have already been answered. This redundant posting of the same question by using different words leads to duplicate question problem. For example if a person asks, "What do you mean by force in physics?" in a forum while in the same forum another question was asked by another person like "Can someone explain force in terms of physics?". Both of these questions are asking for the same answer but are different from each other in terms of the order of the word and syntax. It may happen that one of the question remain unanswered or wait for longer period to get answered as the users of the forums providing the feedback get frustrated by receiving the same question again and again with slight variations. Thus, a good paraphrase detection or duplicate question detection system can classify these two questions and mark them as similar. Therefore restricting the user posting a redundant question by bringing the answers of the questions that are already there in the forum. We implemented 5 deep neural models to detect similar question using neural network architecture. The models correctly identify duplicate questions and mark them as duplicate or not duplicate. These models are independent and do not require any external knowledge, feature engineering and language tools. They use variations of RNN like LSTM and GRU. Siamese architecture allows us to use this RNNs simultaneously and separately for both questions in a pair.

3 Literature Survey

We started our Project by first analyzing the problem statement and the resources already available to us. We thought of an approach which was inspired by the encoder module in [1] to encode the question into a vector and then compute the similarity between these vectors. Finding papers related to our problem statement and approach we chose the first model of Chali et al, [2] for our baseline implementation. To better understand the Siamese Architecture we read [3] where Siamese network architecture was first introduced for signature verification. We first went ahead with the baseline mentioned above but since the results were not satisfactory we tried different approaches and implemented [4]. The model described in [4] by Mueller and Thaygarajan is very similar to the one in Chali et al [2], only the part where the distance is computed between the sentences is changed. We got very significant improvements and the results and reasons are described in the Findings and Analysis section.

4 Research Methods

4.1 Dataset Description

We used the Quora Dataset [1] for our experiments. This dataset originally released by Quora has 400K pairs of questions each marked with a ground truth of either 1 or 0, where 1 denotes duplicate pair of questions and 0 denotes the opposite. Each pair of questions has a unique identifier. Each question pair also has a unique id named as qid1 and qid2. The questions are separated in two columns as question1 and question2, thus (question1, question2) forming a pair. This dataset contains 149,306 positive examples (i.e, semantically duplicate questions) and 255,045 negative examples.

4.2 Data Preprocessing and Training

We preprossessed the dataset removing the punctuations and stop-words. We then processed the Quora dataset to create Train, Test and Validation sets. The dataset consists of 364290 question pairs classified as duplicate or non-duplicate. We used 324290 of these pairs for Training, 40000 for Testing and 40000 for Validation. We have primarily used Siamese networks along with different word embeddings and classification methods for training task. We used Adaptive Moment Estimation ADAM [8] optimizer with a learning rate of 0.001. We used ADAM because of its intuitiveness during the training phase and its less memory requirements. We started with a very small learning rate to avoid over fitting in the early stage of training. ADAM optimizer is adaptive in nature, so learning rates during the training will be set accordingly after each epoch by the optimizer. We tried to minimize the loss function using the binary crossentropy. We experimented on the sequence length using average sequence lengths of the vectors. We set the hidden size of the LSTM and the Bi-Directional LSTM layers as 64 in order to get a more high-level representation. We processed the questions by batch of 2048 question pairs each time for training the network. Typically training is faster with mini-batches as the weights are updated after each propagation. But one of the disadvantages is that the smaller batches provide less accurate estimate of the gradient. So we optimally chose the batch size. The whole training process ran for 20 epochs. Epochs are one forward pass and one backward pass on all the training examples

[1]https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs

4.3 Siamese LSTM

4.3.1 Softmax

The first model that we implemented was a very basic Siamese network which has a single LSTM. Each question in the pair is first embedded using word embeddings like word2vec and then passed through the LSTM. The final hidden vectors are summed together and passed through a soft max activation layer to classify as duplicate or non-duplicate. This model gave us 76.5% accuracy on the test dataset. Figure 1 shows the architecture of the model.

4.3.2 Cosine distance

Our next experiment involved trying out different distance methods. After obtaining the hidden state outputs from the Siamese network, we find the cosine distance between the two vectors. This model gave an accuracy of 63%.

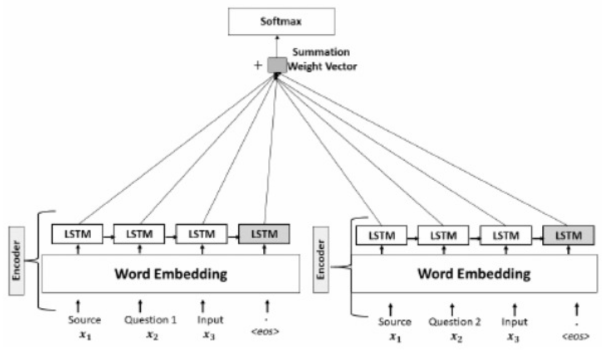


Figure 1: LSTM + Softmax model

4.3.3 Manhattan Distance

Similar to the previous two models, we passed the two sentences through the word embedding layer trained using word2vec, then passed the word embeddings through an LSTM to obtain the sentence context vectors. Then we calculated the Manhattan distance between the two vectors. This model gave a good accuracy of 81.47%. Figure 2 shows the architecture of the model.

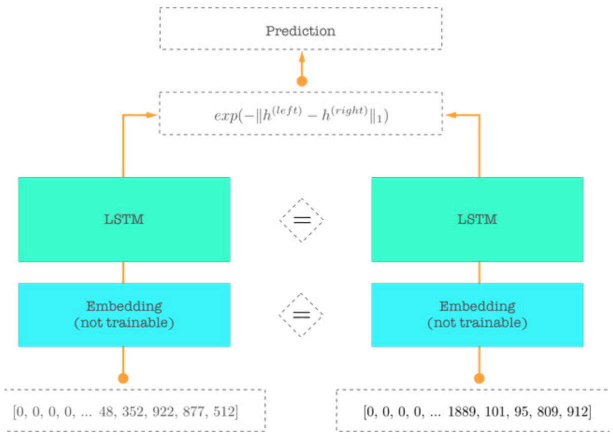


Figure 2: LSTM + Manhattan model

4.4 LSTM with Attention

The encoder learns the hidden representations in the same way as described above. The two output vectors coming form the encoder are passed to an attention mechanism layer. We added a dot attention mechanism layer [9] for it's efficiency and simplicity. The dot attention mechanism is actually the dot product between two hidden vectors. Using attention mechanism removes the bottleneck of processing the fixed length hidden vectors which propagates important information over longer distances. An attention mechanism captures the context of the vectors, keeping in mind that we do not lose information while reducing matrix dimensions. The attention layer adds an extra weight to the vectors. At each time step t, a context vector c_t is generated by the LSTM hidden state(h_i) from the encoder. The weight α_{ij} is the strength of the attention i^{th} word in the target vector to the j^{th} word in the source vector. The general idea behind the attention vector is that it tells us how much focus should be given to a particular source word at a particular time step of the encoder. The larger the value of α_{ij}, the more focus will be given to a particular word while forming the next vector representation.

$$c_t = \sum_{i=1}^{N} a_{ij} \times h_t \tag{1}$$

$$a_{ij} = \frac{exp(h_t^{Q_1} \times h_t^{Q_1})}{\sum_i exp(h_t^{Q_1} \times h_t^{Q_1})} \tag{2}$$

Where, $h_t^{Q_1}$ and $h_t^{Q_1}$ are the LSTM hidden states at each time step t coming from the question 1 encoder and question 2 encoder, respectively. Then we merge the encoder outcome of question 1 with the outcome of the attention layer to form the final representation. We use the same softmax classifier as described in the previous model to mark the questions as duplicate or not-duplicate. This model gave us 83.34% accuracy on test dataset. Figure 3 shows the model.

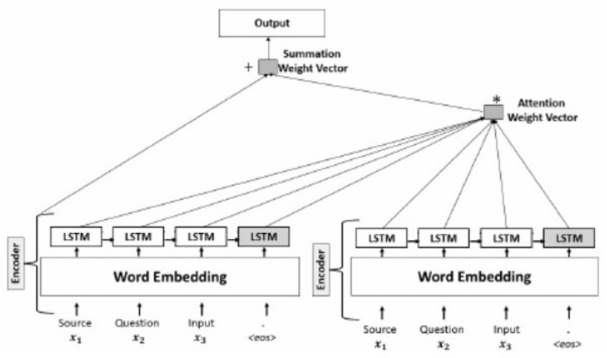


Figure 3: LSTM + Attention model

4.5 Bi-Directional LSTM-Stacking with Attention

We used bi-directional RNN (Bi-RNN) [10] as encoder for learning the hidden states but for

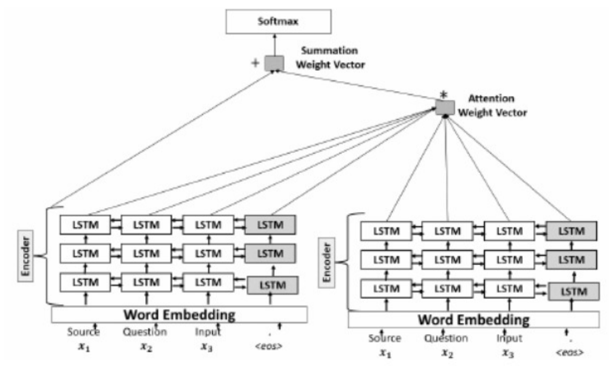


Figure 4: Bidirectional LSTM Stacking + Attention

our case we replaced the RNN with an LSTM making it a bi-directional LSTM (Bi-LSTM). Simple uni-directional LSTM that learns the hid- den annotations h_t at time t as processing the input from left to right is as follows:

$$h_t = f(x_t, h_{t-1}) \tag{3}$$

We implemented Bi-LSTM that processes each ques- tion word embedding representation in both for- ward and backward direction calculating the hid- den annotations h_t at each time step t. For each position t, we merge the hidden states by concatenation to achieve the final hidden state:

$$h_t = \overrightarrow{h_t} \oplus \overleftarrow{h_t} \tag{4}$$

Where operator $\oplus$ indicates concatenation and $\overrightarrow{h_t}$, $\overleftarrow{h_t}$ represents the forward and backward representation, respectively. Forward representation $\overrightarrow{h_t}$ is calculated from $\overrightarrow{h_t}$ = LSTM(x_t,$\overrightarrow{h_{t-1}}$) and backward representation $\overleftarrow{h_t}$ is calculated from $\overleftarrow{h_t}$ = LSTM(x_t,$\overleftarrow{h_{t-1}}$). In this model, we implemented a 3-layer Bi-LSTM following by stacking 3 Bi-LSTM layers on top of each other. A 3-layer Bi-LSTM makes the model deeper and each layer learns and extracts more information of a sentence. Then, we passed the encoder outputs to an attention layer (same attention layer mechanism described in the previous model in section 4.4) to obtain an attention representation vector. We also follow the same merging technique of the previous model to merge the outcome vector of question 1 encoder with the attention representation vector to form the final representation. We use softmax classifier for identifying the classes between duplicate and not-duplicate. In this model, we use binary crossentropy as the loss function. Figure 4 shows the model structure.

4.6 Bi-Directional LSTM-Stacking with Attention-CNN with Maxpooling

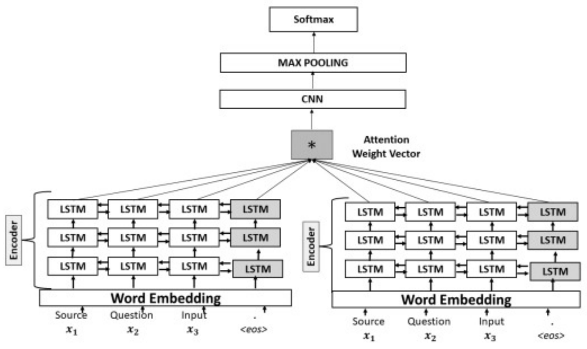


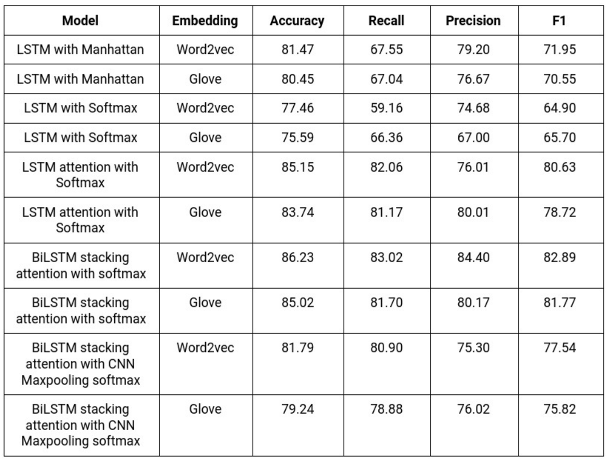
Figure 5: Bidirectional LSTM Stacking + Attention + CNN + Maxpooling

We use three Bi-Directional LSTMs stacked together as the encoder to learn the question representations and then use the same Attention Mechanism as highlighted in the previous models. We add a CNN layer along with max-pooling layer.

We applied a one dimensional convolution layer with a filter size of 5*d along with a stride size of 1, where d represents the dimensionality of the word embeddings. Filter size of 5 allows us to build a 5-gram model for a large sentence to predict and extract information. On top of the CNN layer, we use a max-pooling layer of 2*2 to reduce the matrix dimension generated from the convolution layer. Max-pooling layer is used to extract the crucial local features to form a fixed length feature vectors. Then we use a softmax classifier to classify the questions into duplicate or not-duplicate class. We use Binary Corssentropy as the Loss function.

5 Findings and Analysis

Figure 5 shows various models and their accuracy. We compare the results of our systems in terms of Accuracy, Precision, Recall, and F1 score. From the results of Figure 1 we observed that the complex models performed better than simple ones. The Bi-Directional LSTM-Stacking with Attention model performed best among all other models. Attention over the word vectors is also playing a key factor for



Model	Embedding	Accuracy	Recall	Precision	F1
LSTM with Manhattan	Word2vec	81.47	67.55	79.20	71.95
LSTM with Manhattan	Glove	80.45	67.04	76.67	70.55
LSTM with Softmax	Word2vec	77.46	59.16	74.68	64.90
LSTM with Softmax	Glove	75.59	66.36	67.00	65.70
LSTM attention with Softmax	Word2vec	85.15	82.06	76.01	80.63
LSTM attention with Softmax	Glove	83.74	81.17	80.01	78.72
BiLSTM stacking attention with softmax	Word2vec	86.23	83.02	84.40	82.89
BiLSTM stacking attention with softmax	Glove	85.02	81.70	80.17	81.77
BiLSTM stacking attention with CNN Maxpooling softmax	Word2vec	81.79	80.90	75.30	77.54
BiLSTM stacking attention with CNN Maxpooling softmax	Glove	79.24	78.88	76.02	75.82

Figure 6: Results on Kaggle Dataset

improving the performance. Also we see no significant difference in the accuracy when we change the word embeddings from Word2Vec to Glove. This shows that the model is robust and not dependent on the word embeddings used. Also this confirms that Word2Vec and Glove are equally good.

While running our experiments on the models, we observed a similar pattern for the convergence. The accuracy after the 3rd epoch increased sharply comparing to the previous epochs and became flat over the next 4 or 5 epochs. We observed the similar situation in the case of the training loss too. The loss at the beginning of the training was high but it quickly came down and flattened after 7th and 8th epochs.

We also improved the Softmax model which was earlier giving an accuracy of 40.03% to give an accuracy of 76.53% later. During this process we found the mistake was in the layer's activation function.

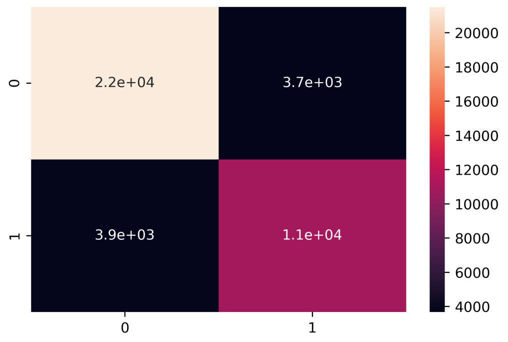


Figure 7: Confusion Matrix for LSTM with Attention

We also researched the cause why Manhattan distance is the best. We found out that using l2 rather than l1 norm in the similarity function can lead to undesirable plateaus in the overall objective function. This is because during early stages of training, l2-based model is unable to correct errors where it erroneously believes semantically different sentences to be nearly identical due to vanishing gradients of the Euclidean distance. We found that utilizing the Manhattan distance outperforms other reasonable alternatives such as cosine similarity.

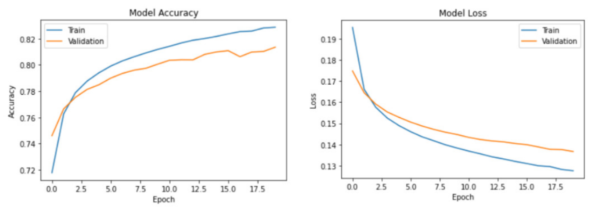


Figure 8: Graphs of BiLSTM + Attn model

6 Limitations

Currently our models are performing fairly well, but the way the attention [6] is used to compute the interdependencies is a limitation which can be improved by adding more complex and better attention layers.

7 Future Scope

We would also like to explore and expand to [5] which gives an improve in accuracy. Also we would like to explore into more depth of how we can through feature engineering, make better models. We would also like to explore how self attention (transformers) [7] will help to get a better representation of the sentence and thereby increasing the performance.

8 References

[1] Ilya Sutskever, Oriol Vinyals and Quoc V.Le. 2014. Sequence to Sequence Learning with Neural Networks. CoRR abs/1409.3215 (2014). `https://arxiv.org/abs/1409.3215`

[2] Yllias Chali and Rafat Islam. 2018. Question Question Similarity in Online Forums. In Proceedings of the 10th annual meeting of the Forum for Information Retrieval Evaluation (FIRE'18). `https://dl.`

acm.org/doi/10.1145/3293339.3293345

[3] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature Verification Using a "Siamese" Time Delay Neural Network. In Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS'93). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 737–744. https://dl.acm.org/doi/10.5555/2987189.2987282

[4] Jonas Mueller and Aditya Thyagarajan. 2016. Siamese Recurrent Architectures for Learning Sentence Similarity. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16). AAAI Press, 2786–2792. https://dl.acm.org/doi/10.5555/3016100.3016291

[5] Arpita Das, Harish Yenala, Manoj Kumar Chinnakotla,and Manish Shrivastava. 2016. Together we stand:Siamese Networks for Similar Question Retrieval. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers. https://www.aclweb.org/anthology/P16-1036.pdf

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. CoRR abs/1409.0473 (2014). https://arxiv.org/abs/1409.0473

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. 2017. Attention is All You Need. CoRR abs/1706.03762 https://arxiv.org/abs/1706.03762

[8] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. CoRR abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

[9] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention- based Neural Machine Translation. In Empirical Meth- ods in Natural Language Processing (EMNLP). Association for Computational Linguistics, Lisbon, Portugal,1412–1421.

[10]Alex Graves, Navdeep Jaitly, and Abdel rahman Mo- hamed. 2013. Hybrid speech recognition with Deep Bidi- rectional LSTM. In ASRU.