Identifying Duplicate Questions

Aniruddh Shetty
aniruds@andrew.cmu.edu

Nishant Gurunath
ngurunat@andrew.cmu.edu

Prerna Patil
pmpatil@andrew.cmu.edu

Abstract

The information available on online user forums is abundant however, very distributed. To increase the efficiency of such forums, it is essential to combine the entire knowledge on one subject together. One way to achieve this objective is to detect semantically equivalent questions and merge them. This is a challenging task as two questions can have different words and sentence structure yet be similar in meaning. This study proposes two deep learning architectures that capture the semantic similarity between questions-pairs on Quora. We implement siamese convolutional neural network (CNN) and recurrent neural network (RNN) architectures and compare their performance to baseline models from literature. An extensive study is performed on variable length word sequences in CNNs and its performance is analyzed with change in kernel size. The siamese architectures prove to be very effective for analyzing sentence pairs. We also look at the RNN architecture and study the comparison between the two architectures.

1 Introduction

On knowledge sharing websites like Quora, stack exchange, reddit, Yahoo answers etc. it is important to identify questions addressing the same topics so that users can access all the information at one place. Currently, the information is highly scattered. This poses a challenge both as a consumer (not knowing where all to look for the right answers) and producer (the same question may have been answered hundreds of times) of the information. Therefore, there is a definite need to coalesce this scattered information to increase the efficiency of such forums.

In this study, we address this issue by developing neural network models to detect duplicate questions. We work on the data-set released by Quora. We propose siamese architectures for shift invariant models such as convolutional neural networks and recurrent neural networks. Siamese architectures enable us to train the same model for both the questions and hence, increase the training rate. We adopt shift invariant models as opposed to regular dense neural networks. This helps to capture the context from different parts of the questions and reduce computation complexity. We implement two deep learning models and their variants to detect semantic similarity as these methods take into consideration paraphrase evaluation and textual entailment.

We start by describing the previous work on this task and introduce our motivation and baseline models for this approach. This is followed by a detailed insight into the Quora dataset. We then present a Siamese CNN model with parallel kernels that captures meaning between different sets

of words at the same time. Then we explain the concept of attention that captures dependence and correlation among the words of the two questions. This is followed by description of a siamese BiLSTM architecture. We then conclude by contrasting the results of different approaches and the methods that can be employed in future to improve these results.

2 Related Work

NON-DL approach . We need carefully designed features for this approach. Thus, we need external resources like Wordnet [Wu and Palmer, 1994], POS(Parts of Speech tagger) or NER(Named Entity Recognition). Wordnet is a heirarchical structure to calculate similarity between a word pair. Different approaches were tried out with the pathlengths of word pairs in Wordnet like averaging the maximum pairwise conceptual scores. These approaches were simple and easy to implement but required external knowledge and required carefully thought out features. This is tackled by using deep learning approaches.

DL approaches. Some crude approaches are summing up the word embeddings to get the resultant sentence vectors[Blacoe and Lapata, 2012] or taking a weighted average of them using the TF-IDF weights, easily implemented using sklearn. But these were very simple approaches and gave mediocre results. Currently, there are mainly two types of deep learning frameworks used for paraphrase detection. The first is based on the Siamese architecture[Bromley et al, 1993], where the same neural network(CNN or RNN) is applied on the two sentences individually to encode both of them into sentence vector in the same embedding space. Then a similarity metric is applied on these vectors to get the final score. The advantages of this framework is the model is smaller and can be trained in lesser time and the sentence vectors can be used for visualization. But they treat the sentences separately, as independent entities. The second framework eliminates this problem by using attention models.[Wang et al, 2016] The features obtained from both sentences are matched and then they are aggregated into a vector which is used for measuring similarity.

We have implemented the Siamese architecture using CNN and RNN. In CNN, we have added 4 different categories of N-gram filters giving us information at different levels of granularity. And then concatenated them to get a final sentence vector. Also, after this, we implemented the attention based model for CNN, which compares the two sentences at each level of granularity and averages the convolution features using the attention weights.

3 Dataset

The raw dataset for this problem is obtained from www.kaggle.com/quora/question-pairs-dataset. The data contains a pair of question with a column 'is_duplicate' informing if the quesiton pair is duplicate or not. An example of the data is given in Table 1

id	qid1	qid2	question1	question2	is_duplicate
245	491	492	How do you get a book published?	What are the good ways to write and publish a book?	0
253	507	508	What are the qualities of a good leader?	What are some good qualities of a leader?	1

Table 1: Example of the questions in raw dataset

Pre-processing is carried out on this raw data set, wherein we have used pre-trained glove vectors for each of our model. We used the mode trained on 6B tokens of Wikipedia data for a vocabulary size of 400,000(uncased) and the word vector dimension is 300. For padding, we have taken the zero vector, so that no matter what the model weights are, these padding will not contribute anything ahead. And for out of vocabulary words, we have taken a single random vector initialized at the beginning. This single random vector is used for all out of vocabulary words because they are a small fraction.
For tokenizing the sentences into words, we just used split and removed unnecessary characters like '?,[]().<>:;' We also tried out the sklearn vectorizer but that gave almost the same result but the execution time increased. Thus, we decided to go with simply splitting the sentences and stripping

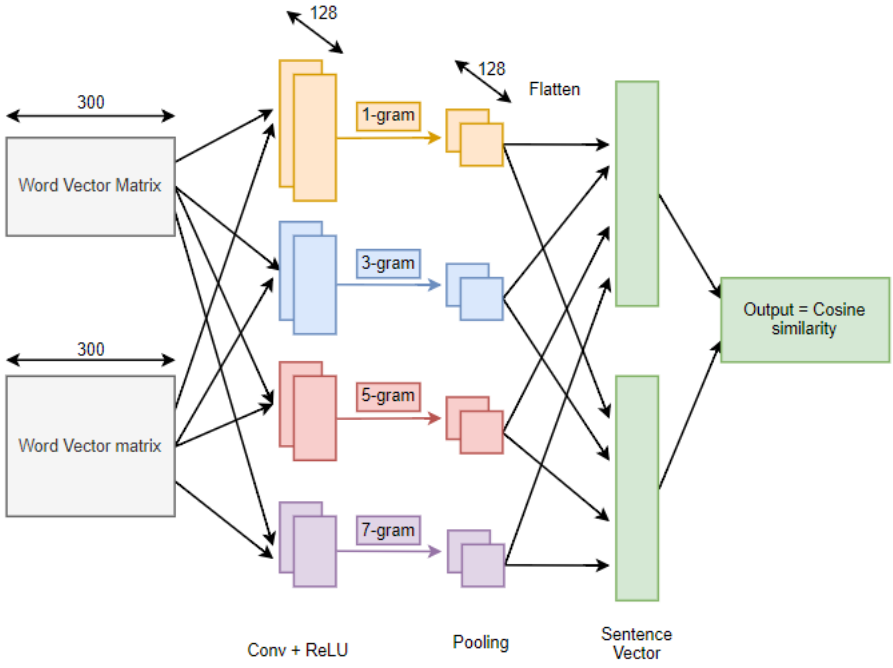

Figure 1: Siamese Convolutional Neural Network

the extra characters.
The dataset given to us was biased. It had almost two-thirds negative examples and one-thirds positive examples. So we took first 5000 positive and 5000 negative examples to make our test dataset. Next 5000 positive and 5000 negative exmaples to make up our validation dataset. And the rest(384,351 question pairs) made up our training dataset.

4 Models

4.1 CNN architecture

We have used a Siamese CNN model which means that both the questions are fed through the same network and we get an output sentence vector for each, on which we apply a similarity metric to get our final output. There are mainly 4 parts in our network as follows

Input Layer: The CNN model takes as input a word vector matrix whose rows represent the word vectors of individual words in a sentence. Hence the number of rows can vary from question to question but the number of columns is equal to the word vector dimension i.e 300. So we get a feature map of dimension q_1 x d_0 where q_1 is the question length and d_0 = 300 is the dimension of the word vectors.

Convolution Layer: To capture the features at different levels of granularity, we have considered 4 different categories of convolution filters namely 1-gram, 3-gram, 5-gram and 7-gram. They represent how many words are considered jointly in a filter. There are 128 filters of each type to learn complementary features from the same word windows. Let the filter width be w. Then each filter will have the weight $W \in \mathbb{R}^{wd_0}$(a vector of w x d_0 dimension). And output will be 128 column vectors as shown. It can also be resahped into a matrix with 128 rows as used in the attention model.

$$O_i = ReLU\left(\mathbf{W}.v[i:i+w-1,:] + b\right)) \tag{1}$$

Here $O \in \mathbb{R}^{q_1+w-1}$is the output of the Convolution layer, v is the input word vector matrix and $b \in \mathbb{R}^w$ is the bias term.

Pooling Layer: This is introduced to take care of the fact that the two questions inputted might have different sizes and thus their convolution layer's output sizes might also vary. Thus we do average pooling and convert the output of each category of filters(N-gram) to a 128 dimensional column vector(a fixed length vector).

Output Layer: From the Pooling layer, we got four(one for each type of the N-gram filters) 128 dimensional vector for one question. We concatenate them to make a 512 dimensional vector. And add a fully connected layer to make the final 256 dimensional sentence vector. We pass both the questions through the Siamese network and get two 256-dimensional sentence (s_1 and s_2) vectors. To get the final score we take the cosine similarity measure between s_1 and s_2.

For training, we calculate the mean square loss between the cosine similarity value and the actual training label. And we used Adam optimizer for updating the weights.

4.2 Attention Model

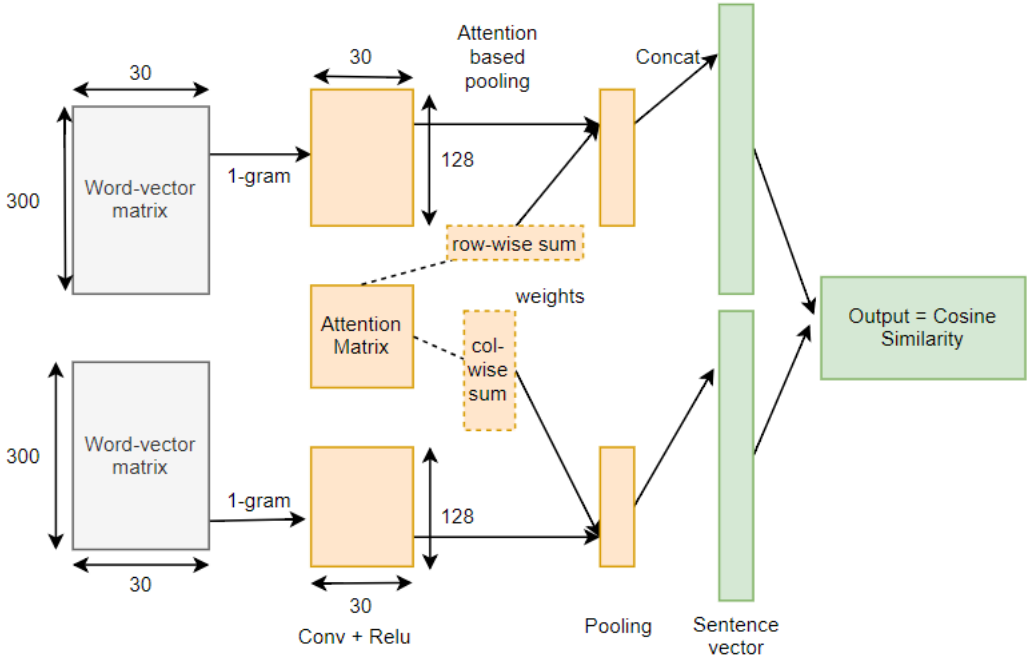

Figure 2: Attention Model

This model is an improvement over the basic Siamese CNN model with 4 different types of N-gram filters. The Siamese model treats the two questions inputted to it as independent but they are not. According to the question pair, there might be words which should be given more weight and some that should be given less weight. So the major update we are going to do is instead of average pooling, we will do a weighted pooling using the attention weights. Also now since we will compare the words between two sentences, we will have to make them of equal sizes with padding. We have selected a question size of 30 based on the length distribution of the questions we had.The figure above shows the flow for 1-gram filters. All the other filters are similar.

We reshaped the output of the the convolution layer to get a matrix of the form 128 x (30 + w - 1) where w is the filter width (1,3,5,7). Here each row represents the output of one of the 128 filters. Let us call each column one unit of the sentence. For 1-gram filters, a unit corresponds to a word, while for 3-gram filters, a unit will correspond to a phrase of 3 words and so on. Now, we'll generate an attention matrix **A** of dimension (30 + w - 1) x (30 + w - 1) comparing all units in q1 to all units in q2. We sum all attention values of a unit (compared with each unit of the other sentence) to get the overall attention weight with which to average pool the unit. As before, we will get a 128 dimensional vector at the output of the pooling layer for each category of filter which we'll concatenate to get a

512 dimensional vector and a fully connected layer will transform it to a 256 dimensional sentence vector.

Let the output of the convolution layer be a feature map F_1 for q_1 and F_2 for q_2. Then the attention matrix A will be calculated as follows.

$$A[i,j] = \text{match-score}(F_1[:,i], F_2[:,j]) \tag{2}$$

A[i,j] entry corresponds to the attention value of the i^{th} column of F_1 and the j^{th} column of F_2. The match-score function can be varied to suit the application. We took it as 1/(1 + $|x - y|$) where $|.|$ is the euclidean distance. A row corresponds to the attention values of one unit of F_1 compared with all units in F_2. Thus to get the attention weights of F_1, we'll do a row-wise sum of the matrix A. And similarly, to get the attention weights of F_2, we'll do a column-wise sum. Mathematically, if $a_{1,j}$ is the attention weight of j^{th} unit of F_1 and $a_{2,j}$ is the attention weight of j^{th} unit of F_2, we have,

$$a_{1,j} = \sum_k A[j,k]$$

$$a_{2,j} = \sum_k A[k,j]$$

$$Op_1 = \sum_j a_{1,j} F_1[:,j]$$

$$Op_2 = \sum_j a_{2,j} F_2[:,j]$$

Here Op_1 and Op_2 are the output of pooling layers for the respective question. An important point to note here is that we did not add any new parameters here over the basic Siamese model but instead we reweighed the output of the convolution layer in such a way that the convolution weights now get adjusted according to the attention values we need for the question pair (like if we want to increase or decrease attention on a particular word depending on the output label)

4.3 LSTM architecture

LSTM (Long short term memory) model is implemented to take care of dependencies in the sentences across the sentence length. While each single sentence is studied using the LSTM, the Siamese architecture takes care of semantic similarity between the two sentences by tying the two weights of each sentence together via the Manhattan distance.

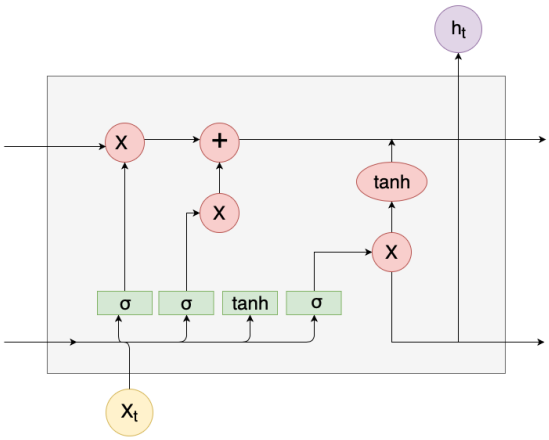

Figure 3: A single layer in the LSTM architecture

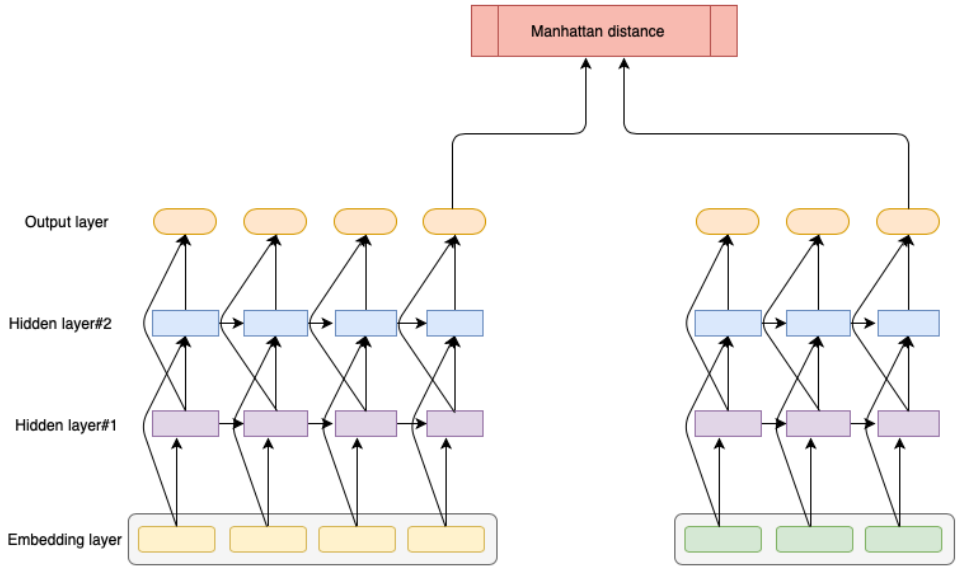

Figure 4: Siamese BiLSTM Architecture

Embedding layer: In this layer, the words are converted to a 300-dimensional vector using the GloVe representation. We follow the same pre-prcessing as used for the CNN part.
Hidden layer:The sentence embedding with padding are fed as an input to the LSTM. A single hidden layer in the LSTM is shown in the figure 3 .

$$i_t = sigmoid(W_i x_t + U_i h_{t-1} + b_i)$$
$$f_t = sigmoid(W_f x_t + U_f h_{t-1} + b_f$$
$$\tilde{c}_t = tanh(W_c x_t + U_c h_{t-1} + b_c$$
$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1}$$
$$o_t = sigmoid(W_o x_t + U_o h_{t-1} + b_o$$
$$h_t = o_t \odot tanh(c_t)$$

Output layer: The Siamese architecture ties the outputs of each of the sentences, using the similarity function given by the negative exponent of the L_1-norm.

$$g(h_1, h_2) = exp(-||h_1 - h_2||_1) \in [0, 1]$$

where, h_1 and h_2 represent the hidden layers in sentence 1 and 2 respectively. Training on this similarity forces the LSTM to capture the semantic similarity between the two sentences.

5 Results

Since our dataset is skewed, two-third negative examples and one-thirds positive examples, a better measure of comparison would be the F1-score which takes into account the recall and precision. Let TP = True Positives, TN = True Negatives, FP = False Positive and FN = False Negatives. Then,

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1-score} = \frac{\text{2*precision*recall}}{\text{precision + recall}}$$

Table 2 and Table 3 depict the variation in performance of the Siamese LSTM and BiLSTM architecture respectively with the network configuration. These experiments were performed to tune the hyperparameters of the model. It is observed that the best results are obtained for the LSTM with 2 layers with 128 neurons in each layer. Increasing the complexity of the network further leads to overfitting.

		Validation			**Test**		
LSTM Layers	No of neurons	Accuracy	Recall	Precision	Accuracy	Recall	Precision
1	128	80.80%	65.10 %	80.80%	82.30 %	80.06%	69.40 %
2	128	81.52%	70.79 %	77.39 %	**82.99** %	71.52%	80.26 %
2	256	81.05%	70.76 %	76.28 %	82.67 %	71.11%	79.88 %
3	256	80.42%	64.78 %	78.47 %	82.28 %	70.37%	79.40 %
3	512	76.60%	48.40 %	80.59 %	79.54 %	63.90%	76.92 %
4	512	76.77%	56.29 %	74.91 %	77.60 %	56.89%	76.31 %

Table 2: Variation in Performance with BiLSTM Configuration

		Validation			**Test**		
Bi-LSTM Layers	No of neurons	Accuracy	Recall	Precision	Accuracy	Recall	Precision
2	128	80.36%	66.35 %	77.31%	**81.85 %**	67.47%	80.12 %
2	256	60.67%	66 %	79.83%	80.12%	60.67 %	80.12 %
3	256	80.33 %	65.13 %	77.44%	81.32 %	68.56%	78.25 %
3	512	77.84%	69.48 %	70.25%	78.78%	61.69 %	71.94 %
4	512	75.18%	44.79%	78.47%	75.18 %	54.19 %	76.46%

Table 3: Variation in Performance with LSTM Configuration

Performance (accuracy) of some of the state of the art models on Quora dataset is shown in Table 4 and Table 5 shows the performance of our Siamese CNN model and its attention variant.

Model	Accuracy(%)
CNN (Baseline)	79.6
LSTM (Baseline)	82.58
GRU	84.95
BiMPM	88.17

Table 4: Performance of Existing Work

Model	Accuracy	Precision	Recall	F1-score
Siamese CNN	81.02	84.13	76.46	80.11
CNN + Attention	81.58	82.07	80.82	81.44

Table 5: Performance of Siamese CNN Architectures

6 Conclusions and Future Work

- Based in this study, we can conclude that Siamese shift-invariant architectures show a great promise for extracting semantic relations between question pairs.
- Our Siamese CNN architecture outperformed the state of the art performance with regard to previous CNN architectures on the Quora data-set.
- Use of attention in text context also proves to enhance the performance over the regular CNN architecture.
- The performance of the proposed Bi-LSTM architecture is still an improvement over the best CNN design and the baseline it is derived from.

6.1 Future work

- We could possibly adopt the bilateral multi-perspective matching design over the existing LSTM architecture and further improve the performance.
- Another method to improve the performance could be using pre-processing vectors which were not trained on the existing set of words. Although we deal with out of vocabulary by using random vectors this might possibly add noise instead of correct semantic reference. This could be a possible area to explore. However, after some literature review, we came to the conclusion that learning embeddings for the out of vocabulary requires a lot of training time and hence, decided to drop it from the current study.
- Further analyses on the raw dataset shows that the performance is dampened by set of misspelled words in the questions. This can be improved by character based models and using those to correct mispelled words.

References

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. ICLR Workshop, 2013.
[2] Bowen Xu, Deheng Ye, Zhenchang Xing. Predicting Semantically Linkable Knowledge in Developer Online Forums via Convolutional Neural Network
[3] Jonas Mueller, Aditya Thyagarajan. Siamese Recurrent Architectures for Learning Sentence Similarity. AAAI Conference on Artificial Intelligence
[4] Jeffrey Pennington, Richard Socher, Christopher D. Manning. GloVe: Global Vectors for Word Representation
[5] Wenpeng Yin, Hinrich Schutze, Bing Xiang, Bowen Zhou. ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs. Computer Science - Computation and Language - 2015.

Online references

- http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Resources

- **GloVe**: https://nlp.stanford.edu/projects/glove/
- **word2vec**: https://code.google.com/archive/p/word2vec/