Comparitive study of machine learning models to identify duplicate questions

Yash Chandarana
ychandar@andrew.cmu.edu

Vivek Pandit
vivekp@andrew.cmu.edu

Ankit Srivastava
ashriva2@andrew.cmu.edu

Abstract

We explored various approaches for the duplicate Quora questions data. We compare hand-engineered features with deep learning methods and show the effectiveness of using a combination of the two. We explore two text embedding methods and different models to combine them including an LSTM and a 1D-CNN. We also show the results obtained using a Siamese LSTM and Bilateral Multi-Perspective Matching.

1 Introduction

The most important challenge faced by Quora, a question answering platform is that of question duplicity in terms of its semantic meaning. This means that different words and grammatical structures may be used to mean the same thing. As a simple example, the queries "What is the most populous state in the USA?" and "Which state in the United States has the most people?" should not exist separately on Quora because the intent behind both is identical. Determining if questions are similar or not can make the knowledge-sharing more efficient in many ways: for example, knowledge seekers can access all the answers to a question in a single location, and writers can reach larger readership than if that audience was divided amongst several pages.

Finding an accurate model that can determine if two questions from the Quora dataset are semantically similar is a challenging task. This is not just a problem faced by the Quora, but by every other question and answering platform. This is a challenging problem in natural language processing and machine learning, and it is a problem for which researchers are always searching for a better solution.

In our current work, we attempt to compare the accuracy of different machine learning models in combination with various feature extraction methods.

In section **Related work** we briefly describe existing works that have been implemented to identify duplicate questions. In **Method** section, we briefly describe various different methods that we have implemented. In the **Results** section, we show the log-loss obtained upon making a submission to the Kaggle challenge. And finally, in **Conclusion, and Future** work section we try to explain what factors may have contributed to changes in accuracy and what can be done in the future to improve the model.

2 Related Work

GLoVe embedding: GloVe [5] is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus. We use the GLoVe embeddings trained on 840 billion tokens of web data available for download from here.

CNN: Kim [2] showed that 1D-CNN alongwith word embeddings can be used for text classification for short text. Given an $l \times d$ input, each filter is $l \times k$ and it only moves in 1 dimension. This helps

Preprint. Work in progress.

the CNN capture all the information in a k-word window. Since the questions presented here are not too long, we can use the same for feature extraction with the above mentioned GLoVe embedding.

Siamese Network: Siamese network was first proposed by Bromley, Jane, et al. [1] . They proposed the Siamese architecture for the signature verification task. They showed significant improvement over the conventional neural network models. This architecture was then utilized for various purposes such as video surveillance, image recognition, and natural language processing. Mueller, Jonas, et al. [4] proposed the use of Siamese LSTM network for learning sentence similarity. They had used Manhattan distance for finding similarity between two sentences. They showed significant improvement in accuracy compared to the state of the art architectures. We have used this Siamese Networks along with our own feature extraction method.

3 Dataset

In addition to the dataset provided, we obtained test data from this Kaggle challenge. We chose to evaluate our methods using the log-loss obtained upon making a submission to the same Kaggle challenge. This gave us a better idea of how good the model was doing and since the test data contains over 2 million question pairs, we believe that the results obtained using this as a test set are more reflective of our model's performance than dividing the given data into train/dev/test. Since the test data did not have actual labels, only log loss values are reported for all the models.

4 Methods

4.1 Baseline

For a baseline model, we decided to use hand-designed features. These features were mostly based on syntactic similarity but as it turns out, we could improve our results for the deep learning based methods by including these features as well. In fact, our best results include these methods too. Table 1 describes the various features used. We group them into three categories:

- **Feature 1**: These are very simple features and most of them are counting based.
- **Feature 2**: These features are based on token-wise similarity and edit distances.
- **Feature 3**: These features look at the character-level N-gram similarities between the two questions.

Since it is expensive to calculate these features, we compute them once for all train, validation and test data and store them on disk, loading as and when required.

We train multiple classification methods using these and also perform ablation studies. This is discussed in Section 5.1.

Table 1: Features used

Feature 1	length of questions, difference in lengths, number of distinct characters, number of question marks, number of capital letters, number of tokens, number of common tokens
Feature 2	Q ratio, W ratio, token set ratio, token sort ratio, partial token set ratio, partial token sort ratio
Feature 3	character level N-gram similarity with N = 3, 4, 5, 6

4.2 Siamese Manhattan LSTM

Siamese network is a class of network architectures that contain two or more identical subnetworks. They have the same configuration with the same parameters and weights. These subnetworks can be Convolution neural networks, or a Simple Neural network. The proposed Manhattan LSTM (MaLSTM) model by Muller et al. [4] uses LSTM networks as sub graphs as outlined in Figure 1. There are two networks , LSTM_a and LSTM_b that are completely similar in architecture, and they take in each question as input to the network.

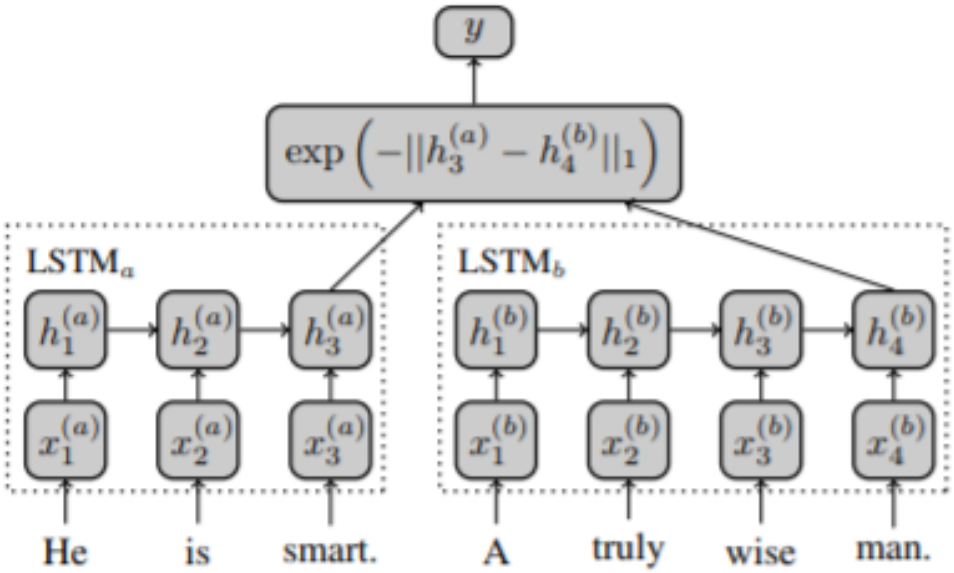

Figure 1: Proposed architecture of Siamese Manhattan LSTM [4]

4.2.1 Feature Extraction of Sentences

Before we could train the Siamese network we need to extract features of each sentence and represent them in the form which Siamese network can understand. For this, we used a pretrained googlenews word2vec model. It includes word vectors for a vocabulary of 3 million words and phrases that they trained on roughly 100 billion words from a GoogleNews dataset. Here each word is represented by a vector of $\mathbb{R}^{300}$ dimension.

First we performed sentence cleaning by removing all stop words (such as a, the, what, why etc.), symbols (such as @, #, $, etc.), punctuations, converting numbers to string values, converting mathematical expressions such as '$1 + 2$' to 'one add two'. After that, we found that there were 160 thousand words in our training dataset which were not trained in googlenews word2vec model. Instead of ignoring them we decided to represent them in some other way. Since googlenews word2vec model gives a vector of $\mathbb{R}^{300}$ dimension, we decided to increase its dimension by one and represent each word by a vector of $\mathbb{R}^{301}$ such that the value of the last row is 0 if the word is present in google news. Now to represent words missing from googlenews we used a bag of word method where we first created a vocabulary of words that were missing from googlenews vocabulary and then assigned each word a normalized index i.e. (index) ÷ (size of missing word vocabulary). Then we concatenated this value to a $\mathbf{0}$ vector of $\mathbb{R}^{300}$ dimension, making it $\mathbb{R}^{301}$ dimensions.

After representing each word from our data with a vector (usually called as embedding) we then represented each sentence using a matrix of size $\mathbb{R}^{mx301}$, where m is the length of the biggest cleaned sentence. Now we pass these embeddings to our Siamese network for training.

4.2.2 Training Siamese Model

The LSTM networks now take in the sentence representation of dimension $\mathbb{R}^{301}$ and transfer it into a representation of $\mathbb{R}^{rep}$. Here we chose $rep = 50$. Then this $\mathbb{R}^{50}$ dimension representation of sentence from both LSTMs is then passed into Manhattan distance formula (Shown in Fig. 1) to check their similarity. Weights of the LSTMs are updated after comparing the output from the similarity distance and actual value (whether similar or not, 0 for nonsimilar and 1 for similar). Details of update equations can be found in Muller et. al. [4]

We used Keras packages to call word2vec model, and implement the Siamese network model.

4.3 Bilateral Multi-Perspective Matching

The Bilateral Multi-Perspective Matching approach, introduced by [6], is a unique method of matching two sentences and predicting the probability of them being semantically equivalent. This method comes under the matching aggregation framework, wherein smaller units (such as words or contextual vectors) of the two sentences are firstly matched, and then the matching results are aggregated (by a CNN or a LSTM) into a vector to make the final decision. The new framework captures more interactive features between the two sentences, therefore it acquires significant improvements.

Fig. 2 shows the architecture of the BiMPM model. The first layer is the word representation layer, wherein a pretrained word2vec model is used to create word embedding vectors for all the words in both the sentences. To include contextual representation of the words, a context representation layer is formed, which utilizes a bi-directional LSTM to generate encodings for each word representation. The motivation behind using bi-directional LSTM is because contextual information of a word can be extracted from both the forward and backward direction in a sentence.

Matching layer uses a cosine similarity function to match the sentences, say P and Q in both the directions, i.e each element of the context representation of P is matched against every time step of Q and vice versa. Two vectors are generated, one for each direction of matching. The aggregation layer is used to create a fixed length vector, again using bi-directional LSTM. The matching function uses multiple weight vectors corresponding to multiple perspectives. This helps in knowing which word representation is closest to all other words in the second question.

Prediction layer consists of two layer feed forward neural network to take in the fixed length vector and uses softmax to predict the probability of both the sentences being semantically equivalent. This model helps in granular matchings between sentences such as phrase to sentence, etc. Also, the matching is done in both the direction, thereby capturing all the semantic equivalences possible, while also maintaining explicit interaction between the two sentences during the encoding procedure, while helps it gain some information.

4.3.1 Implementation

The entire model is trained end to end. The code for this has been released by the authors of [6], which has been used for this project. It uses tensorflow for implementing the machine learning models. Since this is a huge model and computationally very exhaustive, only the final log loss value has been calculated and presented in Table 3. Cross entropy is used as a loss function, and the model is trained using the ADAM optimizer.

4.4 GloVe Embeddings with LSTM, CNN and baseline features

Above is an architecture for this method. We got the GLoVe embeddings for all the words in questions and processed them in three ways:

- LSTM: All the word representations were processed by an LSTM with 256 hidden units. We use the hidden representation at the last cell as a feature representation.
- 1D CNN: We also pass the matrix of word representations through a 1D CNN of kernel size 11 with stride 1 and no padding. We use 256 filters and use global average pooling to get a feature vector of size 256. We use leaky ReLU with $\alpha = 0.3$.
- We also take a mean of all the vector representations.

The above features are extracted for both the questions in a pair and concatenated with the baseline features. This vector is then passed through 5 fully connected layers of size 256 with leaky ReLU ($\alpha = 0.3$) activation. We use a dropout rate of 0.8 and batch normalization after each fully connected layer. The final representation is then passed though another fully connected layer of 1 unit and use sigmoid activation to get a value between 0 and 1. We train this model on the cross entropy loss.

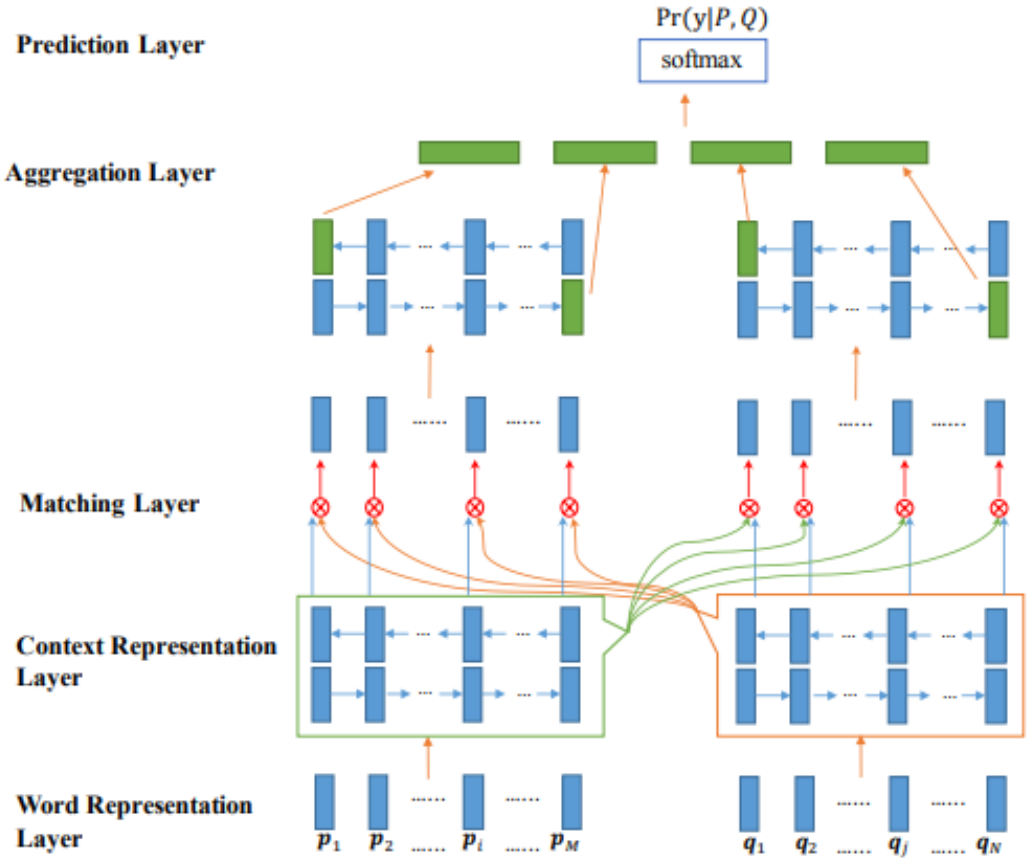

Figure 2: BiMPM Architecture [6]

The above model is trained with Adam optimizer with a learning rate of 0.001. However, the fine-tuning for embedding is trained with a learning rate of 0.0003 with SGD optimizer.

5 Results and Discussions

5.1 Baseline

The first part of table 3 shows results obtained using baseline features with different classifiers.

Since XGBoost worked the best, we tried to study how much each feature set mentioned in table 3 works. A majority vote approach, which predicts a constant probability of 0.37 for each sample in the test setachieves a log-loss of 0.55. Table 2 shows the results improvement as each of the feature is added.

Table 2: Ablation study

Features used	Log loss on test set
None	0.55
Feature 1	0.48
Feature 1 + Feature 2	0.42
Feature 1 + Feature 2 + Feature 3	0.38

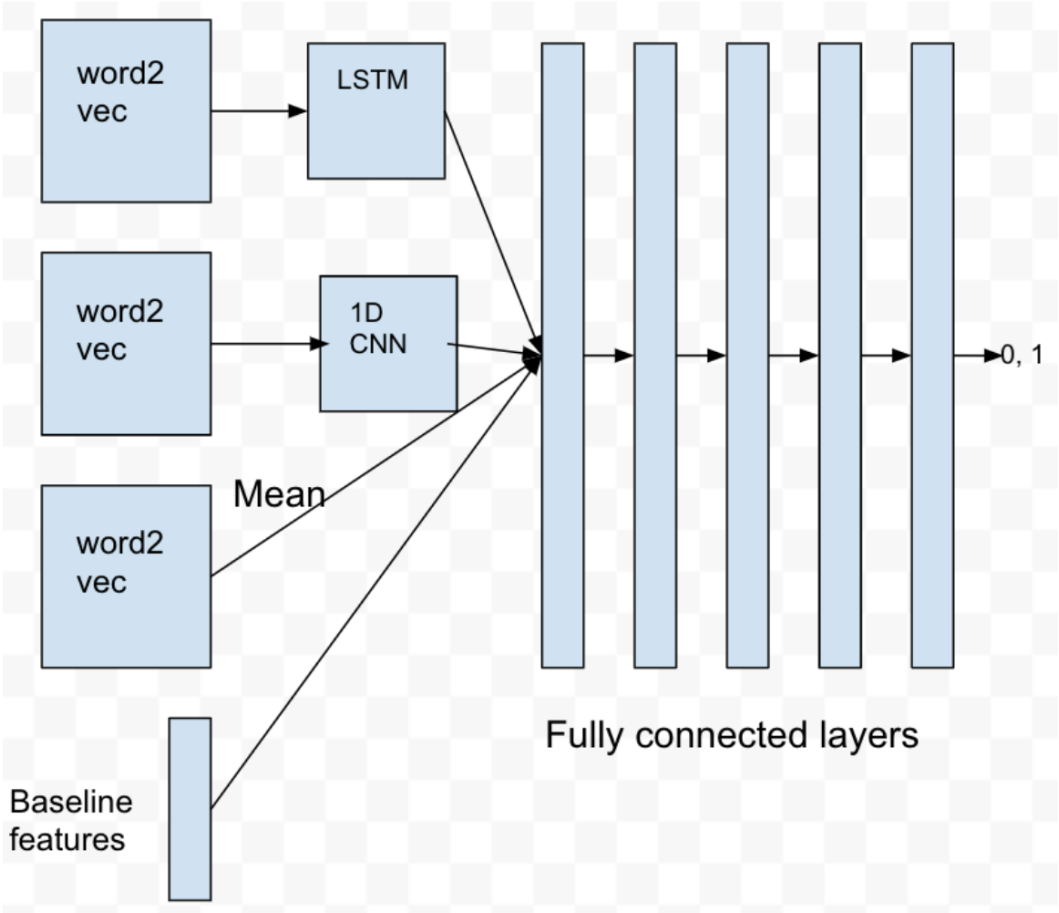

Figure 3: training and validation plots for Siamese Manhattan LSTM

5.2 Siamese Manhattan LSTM

As shown in Figure 4, we ran for about 25 epochs and achieved training accuracy of about 83.7% and validation accuracy of about 82%. The training log loss was about .113 .We then used our trained model to predict the similarity over the test data set. We obtained log loss of about .40 as shown in table 3. We believe the reason for such high loss as compared to other model is due to the way we tried to include the missing word in embedding. Seems like instead of adding extra information it added noise to the features representing sentences. We believe a better way of including those missing vocabulary would have been by retraining word2vec model along with missing words

5.3 GloVe Embeddings with LSTM, CNN and baseline features

As can be seen in Table 3, adding baseline features shows a notable improvement for this method. However, as shown by Major et. al. [3] task specific embedding can work better than general embedding. So, we tried to finetune the GLoVe embedding at a lower learning rate than the rest of the model. This setting gave us the best results among all our experiments.

6 Conclusion and Future Work

Deep learning models improve their performance by using additional baseline features. The motivation behind using Siamese LSTMs was to measure the similarity between two questions, which is passed through the same networks and finally combined using a loss function. The BiMPM model utilizes bi-directional LSTMs to match every contextual embeddings of the words in the questions and then aggregated to then be classified as similar or dissimilar. The intuition behind using this

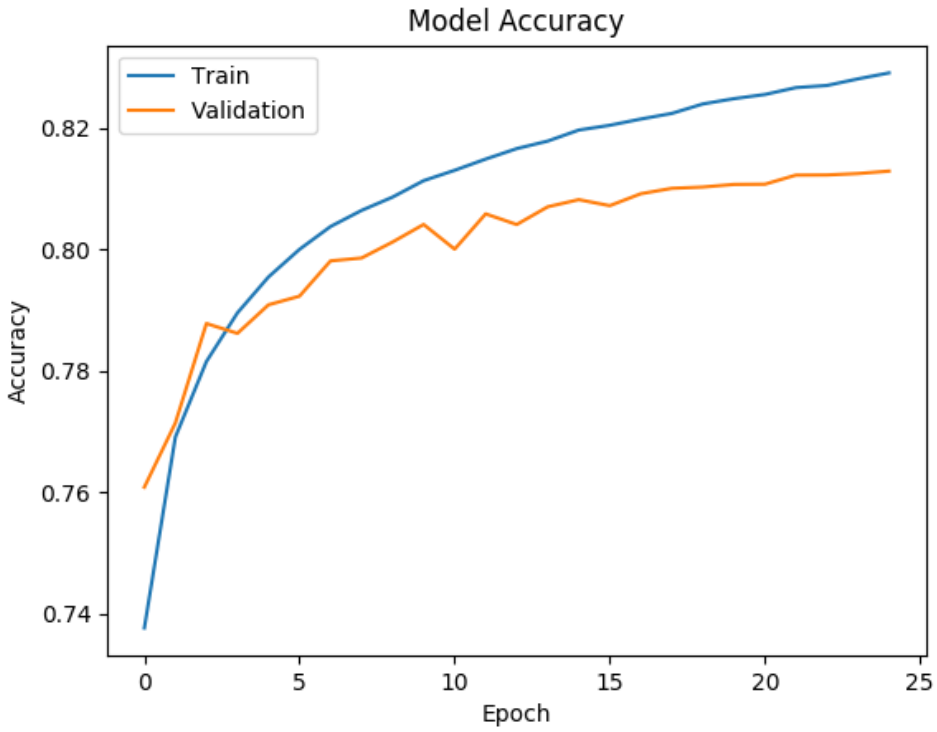

Figure 4: training and validation plots for Siamese Manhattan LSTM

Table 3: Final Results

Classifier	log-loss on kaggle test set
Logistic regression	0.51
Random forest	0.50
SVM	0.41
Adaboost (with Decision trees)	0.41
XGBoost	0.38
Siamese Manhattan LSTM with modified featured	0.40
BiMPM	0.34
GloVe + LSTM + CNN	0.29
GloVe + LSTM + CNN + Baseline features	0.27
GloVe (finetuning) + LSTM + CNN + Baseline features	**0.21**

architecture was to encode contextual information of the words in both the directions and match them against other representations in the second question in both the direction. Also finally, task specific embedding help in improving the performance for the said task. Hence a form of finetuning pre trained models helps in adapting them to a particular instance.
A good direction of future work may involve using transfer learning and experiment with more complex, deeper architectures as well as ensembles of different approaches. We hope these examples illustrate several different approaches to the duplicate question detection problem and provide insights into the performance of different models.

References

[1] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744, 1994.

[2] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[3] Vincent Major, Alisa Surkis, and Yindalon Aphinyanaphongs. Utility of general and specific word embeddings for classifying translational stages of research. *CoRR*, abs/1705.06262, 2017.

[4] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. pages 2786–2792, 2016.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[6] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. *arXiv preprint arXiv:1702.03814*, 2017.