# Quora Duplicate Questions Prediction

**Apoorv Kulshreshtha**
ak3963@columbia.edu

**Samarth Tripathi**
st3029@columbia.edu

## Abstract

Quora recently released a corpus of 3 million question pairs with the labels as duplicate or not duplicate. This is the largest paraphrase corpus currently available. They have also hosted a competition on Kaggle for the same. The challenge is to detect if 2 questions are duplicate, so that Quora can merge the answers to the two separate questions into one, thereby enhancing user experience. Quora currently uses random forests approach for predicting duplicate questions. As this is a classic paraphrase detection problem, and Deep Learning has shown promising results in this field, our project is focussed on exploring the efficiency of various Deep Learning techniques for this task

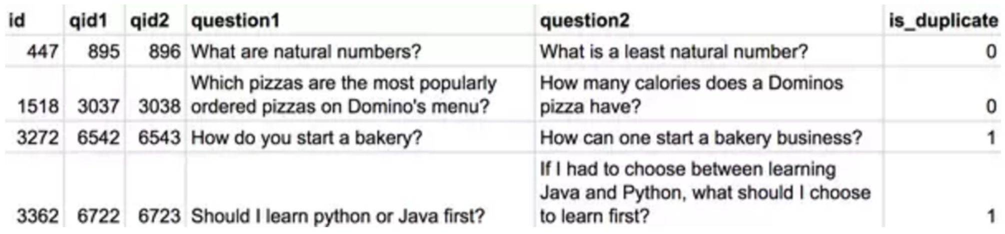
## 1 Introduction

Quora is a question-answering platform with millions of users asking thousands of questions daily. In order to build a high quality knowledge database, it is important that they ensure uniqueness of questions. This would ensure a better user experience as it would help find all the answers to a question at the same place. In NLP domain, this problem is termed as paraphrase detection. In past, a multiple number of other datasets like Microsoft Research Paraphrase Corpus[1] and Twitter Paraphrase corpus[2] have been released, but it is the first time that the number of instances in the dataset is this large. In the following sections, we describe the dataset and the problem which Quora has released.

### 1.1 Dataset

The dataset that Quora released initially, consists of 404351 question pairs [3]. Later, they released another test dataset on Kaggle with 2345806 question pairs. This test dataset contains questions from Quora as well as machine generated random question pairs. Kaggle did this so that people don't manually go over the dataset and submit results, which would be a policy violation for the challenge. The train dataset consists of 255045 non-duplicate (negative i.e. 0 is-duplicate column entry) and 149306 duplicate (positive i.e. 1 is-duplicate column entry) question pairs. Hence, we see that there is a class imbalance in the train data. In the later sections, we will describe how we biased our model accordingly to give better predictions and lower log loss. The shortest question is 1 character long,

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.



| id | qid1 | qid2 | question1 | question2 | is_duplicate |
|---|---|---|---|---|---|
| 447 | 895 | 896 | What are natural numbers? | What is a least natural number? | 0 |
| 1518 | 3037 | 3038 | Which pizzas are the most popularly ordered pizzas on Domino's menu? | How many calories does a Dominos pizza have? | 0 |
| 3272 | 6542 | 6543 | How do you start a bakery? | How can one start a bakery business? | 1 |
| 3362 | 6722 | 6723 | Should I learn python or Java first? | If I had to choose between learning Java and Python, what should I choose to learn first? | 1 |

Figure 1: Glimpse of the dataset format

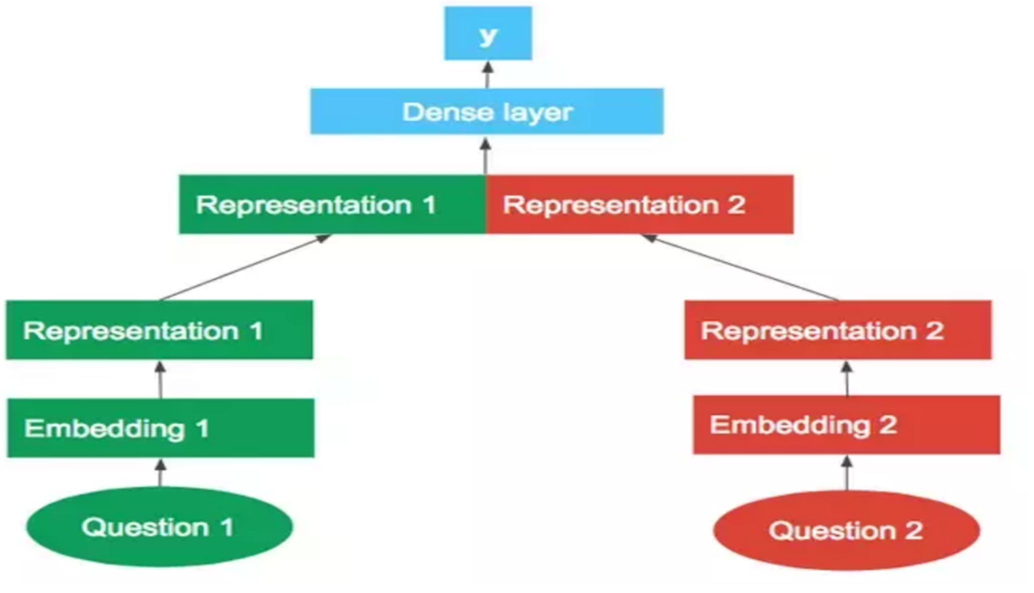


Figure 2: Architecture of Quora's first approach "LSTM with concatenation"

which seems more like a typo. The longest question is 1169 characters long. We remove the instances which have less than 10 characters, as they don't seem helpful. The dataset consists of question ids for both the questions which don't seem informative for our task. A glimpse of the dataset format can be seen in Figure 1.

### 1.2 Problem Definition

In a formal setting, the duplicate question detection problem can be defined as: Given a pair of questions Q1 and Q2, train a model that learns the function f(Q1,Q2) –> 0 or 1, where 1 represents that Q1 and Q2 have same meaning and 0 otherwise.

A trivial duplicate detection approach is to use word-based comparisons and standard information retrieval measures like TF-IDF, to classify the question pairs with an above threshold similarity score as duplicates. However, detecting duplicates based on intent and semantics is more nuanced. For example, if Q1: "What is the capital city of Italy?" and Q2: "What is the capital city of India?", then the usual word based scoring measures will give a very high score to such question pairs, whereas the intent behind these pairs is clearly very different.

## 2 Related Work

Considering the recent performance of Deep Learning in the field of paraphrase detection, a group at Quora picked up the issue of duplicate questions existing on their platform, and explored various deep learning methods during their hack week [4]. Their main motivation was to avoid iterative and computationally complex feature engineering which they used in production systems. In their first approach, they used LSTMs with concatenation to get a vector representation for the two questions. They then fed this vector representation into a dense layer to produce the final classification result. The architecture for the same is described in Figure 2. They achieved an accuracy of 0.87, precision of 0.88 and a recall of 0.86 using this model.
In their second approach, they again used an LSTM to get the vector representations of the 2 questions, but instead of concatenating both of them, they implemented two handcrafted features using them: 1) Distance, which is calculated as sum of squares of the difference of the two vectors, and 2) Angle, which is calculated as an element-wise multiplication of the two vector representations. The architecture for this approach is described in Figure 3. With this approach, they achieved an accuracy of 0.87, a precision of 0.83 and a recall of 0.94.

In their third approach, they used an attention based mechanism similar to machine translation. They trained an attention model for all pairs of words in the two questions, compared aligned phrases and finally aggregated comparisons to get duplicate/not-duplicate classification. The architecture for this approach is described in Figure 4.
Apart from the initial efforts by the teams at Quora, Eren Golge explored the performance of using different word embeddings, with the Siamese Network Architecture, on his blog [5]. Siamese Network is a two way architecture which projects the data into a space, such that similar objects lie closer to one another and dissimilar objects are dispersed over the learned space. Among the approaches that he tried, the first one was to train his own embeddings using Gensim and then use it with Siamese Network. The accuracy he achieved for this model was 0.69. In his second model, he used pre-trained Glove embeddings from Spacy [6] along with the Siamese Network and achieved an accuracy of 0.72. Finally, he used TF-IDF scoring along with Spacy's pre-trained Glove embeddings along with the Siamese Network to achieve an accuracy of 0.79.

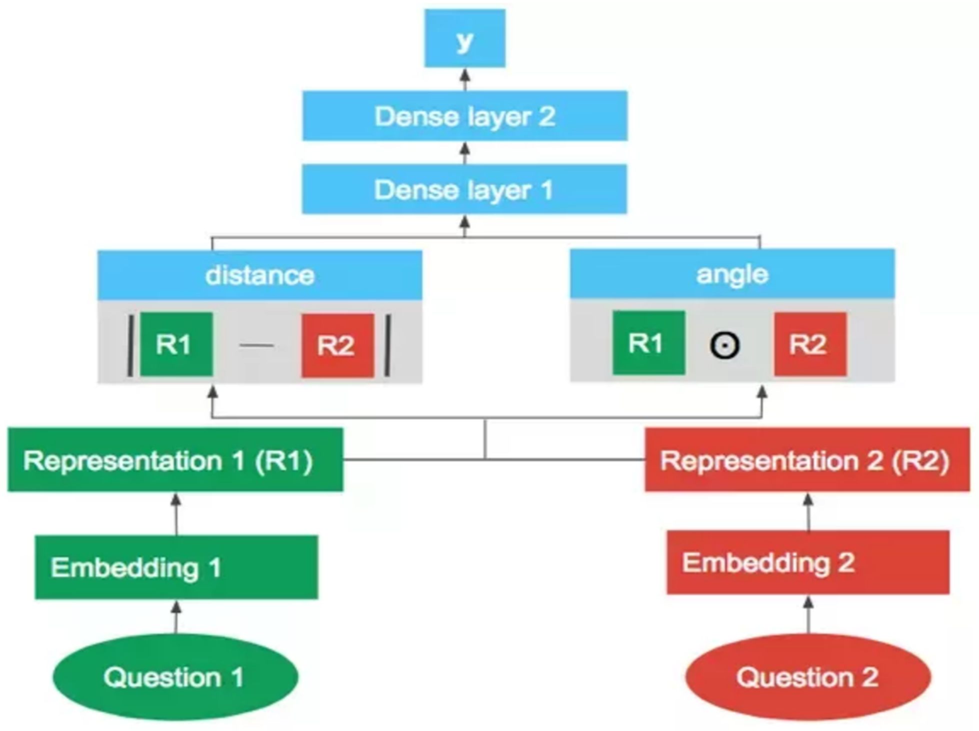


Figure 3: Architecture of Quora's second approach "LSTM with distance and angle"

## 3 Our preliminary models and their corresponding results

After the works mentioned in the previous section came out, Quora launched a Kaggle competition for this challenge with a train dataset consisting of 404302 question pairs and a test dataset consisting of 2345806 question pairs. As described in the dataset section, the test dataset consisted of machine generated random question pairs in addition to the question pairs from Quora. Therefore, for our initial results, we resorted to testing the efficiency of models, described in previous section, on this larger test dataset. Our implementation of Quora's first approach, LSTM with concatenation gave us an accuracy of 0.82 on the validation set. The predictions from this model resulted in a log loss of 0.929 on Kaggle and a corresponding rank of approximately 1200. We then tested the LSTM with attention approach and were able to achieve a validation accuracy of 0.826 and a log loss of 0.57 on Kaggle. This moved our ranking from 1200 to approximately 800. We then experimented with Siamese Networks along with pre-trained embeddings. Our Siamese Network with Glove embeddings implementation resulted in a 0.568 validation accuracy. Same model architecture with word2vec embeddings resulted in a 0.675 validation accuracy. As these performance measures were quite low,

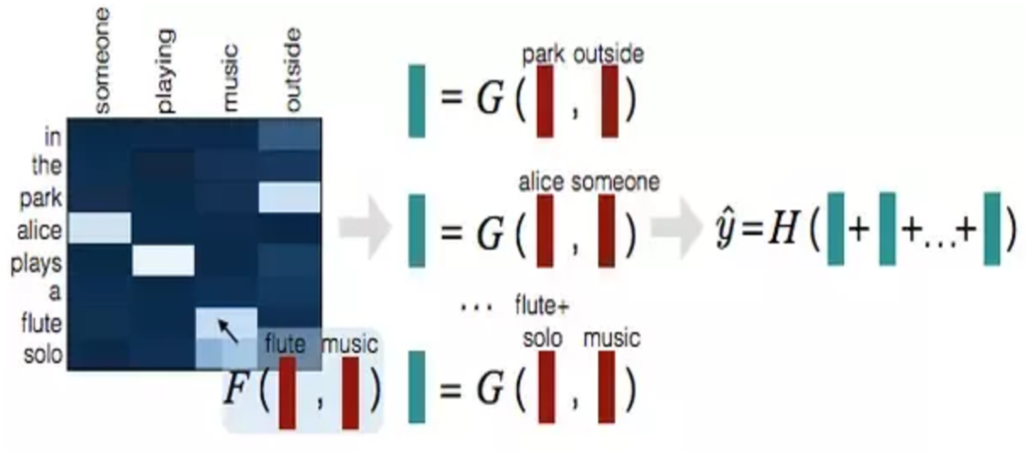


Figure 4: Architecture of Quora's third approach "LSTM with attention"

we didn't use the trained models for making prediction and subsequently getting the log loss on Kaggle. As the LSTM approach gave us promising results, we explored using LSTMs along with incorporating feature engineering. We used the following set of features for our models: [*]

- Unigram common ratio : Number of unigrams common to both the questions / Total number of unigrams
- Bigram common ratio : Number of bigrams common to both the questions / Total number of bigrams
- Trigram common ratio : Number of trigrams common to both the questions / Total number of trigrams
- Length of each question
- Difference in length of the questions
- Number of words in each question
- fuzzy QRatio *: Quick ratio comparison between two strings
- fuzzy WRatio *: A measure of the sequences similarity
- fuzzy Partial Ratio *: Ratio of the most similar substring
- fuzzy Token Set Ratio *: Tokenize the questions, split the tokens into two groups: intersection and remainder. Then construct two strings of the form <sorted intersection> <sorted remainder>. Then take ratios of those strings.
- fuzzy Token Sort Ratio *: A measure of the sequences' similarity, but sorting the tokens before comparing.
- fuzzy Partial Token Set Ratio *: Similar to token set ratio but with substrings of the question instead of the complete string
- fuzzy Partial Token Sort Ratio *: Similar to token sort ratio, but with substrings of the question

Using these features we were able to achieve an improvement in validation accuracy as well as log loss. The validation accuracy for this model is 0.836 and the log loss is 0.283. It also helped us significantly improve our ranking on Kaggle, with the rank being in lower 200s.
These were some of the preliminary models that we tried and their corresponding results. In the following sections we will discuss our final approach along with our best log loss model and our best validation accuracy model.

[*] As implemented in Python FuzzyWuzzy library: Fuzzy String Matching in Python [7]

## 4 Our Final Approach

In this section we discuss in detail two of our approaches that gave the best results. Both models represent an ensemble based approach using various Neural Network paradigms, and are complemented with contemporary machine learning techniques and standards like BatchNormalization, Dropout, Rectilinear Units etc. Their architectures are also elaborated, and are implemented using Keras[8] and Tensorflow[9] Libraries in Python.

### 4.1 Our best Log Loss Model

Our best log loss model is an ensemble built using a combination of stacked LSTM layers, different word embeddings, Dense layers and text based features. We first feed our pair of input sentences to a feature extractor which generates text based features same as the ones mentioned in the previous section. We feed the outputs of the feature extractor to a Fully connected Dense layer of 150 neurons, followed by BatchNormalizing the outputs and applying Dropout.

We then apply pre-trained word embeddings to our input sentences. Our first model uses Glove[11] based embedding, and our second model uses the same architecture, but with Word2Vec[10]. Glove uses an unsupervised algorithm to obtain vector representation for words based on word-word co-occurrence. Word2vec on the other hand, uses 2 layer deep network and generates word embedding by trying to predict every next word using softmax and minimizing the loss. The embedding based sentence vectors are next fed into a LSTM[12] units each. LSTMs are a form of recurrent neural networks which handle the vanishing and exploding gradient problems efficiently using cell state and extra gates for storing context over long sequences. LSTMs have recently become extremely popular for analyzing text based sequences. In our model, the LSTM layer consists of 250 units and returns the full sequences of embeddings. The LSTM outputs for both the LSTM layers (one for each question) are then merged with concatenation and fed to another LSTM layer, again with 250 units. The final output of the second LSTM layer is then fed to a fully connected dense layer with 150 neurons, followed by BatchNormalization and Dropout.

Finally, the output of all three Dense layers (one following feature extractor, and two following stacked LSTM models each for Glove and Word2vec) are merged with concatenation. This concatenated representation is now fed to another another Fully Connected Dense layer with 150 units followed by again by BatchNormalization and Dropout. Finally, the output is fed to a single neuron with Sigmoid activation to allow for a probabilistic prediction of how close the two questions are from a paraphrase perspective. The network uses Adam[13] as optimizer and Binary Cross-Entropy as the loss function. Adaptive Moment Estimation or Adam is an optimizer that computes adaptive learning rates for each parameter. It uses exponentially decaying average of past gradients and past squared gradients and generally converges faster than traditional optimization techniques. For our implementation we use a consistent Dropout value of 0.25, Dropout[14] is a highly efficient regularization technique of dropping units in a network to prevent overfitting. All the Dense layers, except the final neuron, uses ReLu as the activation function. Activation functions are used to transform the activation level of a neuron into an output signal and are generally non-linear. Moreover ReLu has been argued to be more biologically plausible and practical. The model is detailed in Figure 5, where BN stands for BatchNormalization and Dr stands for Dropout.

### 4.2 Our best validation accuracy model

Our best Validation accuracy model is an ensemble built using a combination of many different models. We first feed our pair of input sentences to the same feature extractor discussed previously which generates text based features . We also use similar two LSTM based models as before, but instead of each being a 2 layered stacked LSTM network, we use a single LSTM model. Our first model uses Glove based embedding on each question followed by an LSTM layer with 250 units. Then we merge the final LSTM embeddings of both questions and pass the output to a Dense layer with Batch-Normalization and Dropout. Our second model uses the same architecture, but with Word2Vec Embeddings.

For our third model we use 1D Convolutional Neural Networks with feature lengths of 3 and 128 kernels over each question. The layer creates a convolution kernel that is convolved with the layer input over a single spatial dimension. We also add Batch-Normalization and Dropout with this layer

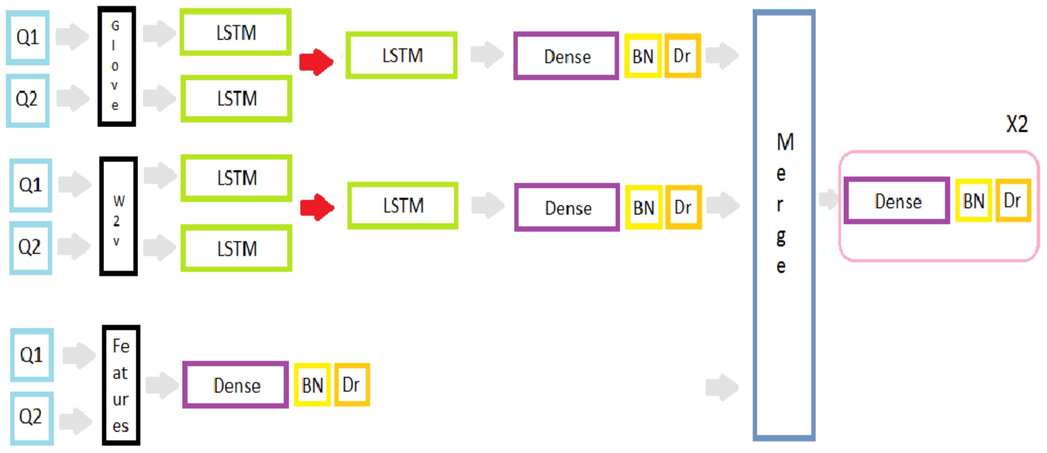


Figure 5: Our Best Log Loss model

and repeat this substructure once more. We then flatten the outputs for both the question and merge them with concatenation. We then add a Dense layer of 150 neurons with Batch-Normalization and Dropout to the output. Our fourth models uses a Time-Distributed Dense layer of 250 neurons followed by BatchNormalization and Dropout. Time-Distributed layers allow us to apply the same Dense layer on every temporal slice of one dimension, allowing it to handle sequences. We take the outputs for both the question and merge them with concatenation. We then add a Dense layer or 150 neurons with Batch-Normalization and Dropout to the output. We finally merge the outputs of all the four models and the model from feature extractor with concatenation.

This concatenated representation is now fed to a substructure of Fully Connected Dense layer with 150 units followed by Batch-Normalization and Dropout, which is repeated twice. Finally, the output is fed to a single neuron with Sigmoid activation to allow for a probabilistic prediction of how close the two questions are from a paraphrase perspective. The network also uses Adam as optimizer, Binary Cross-Entropy as the loss function, consistent Dropout with probability of 0.25 and ReLu as the activation function for Dense and Convolution layers. The model is detailed in Figure 6, where BN stands for BatchNormalization and Dr stands for Dropout.

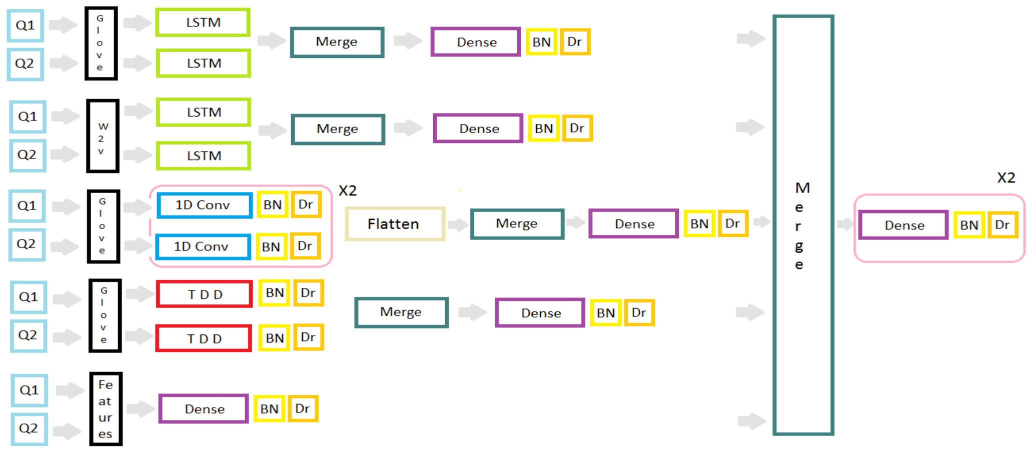


Figure 6: Our best validation accuracy model

## 5 Model tuning and other improvements

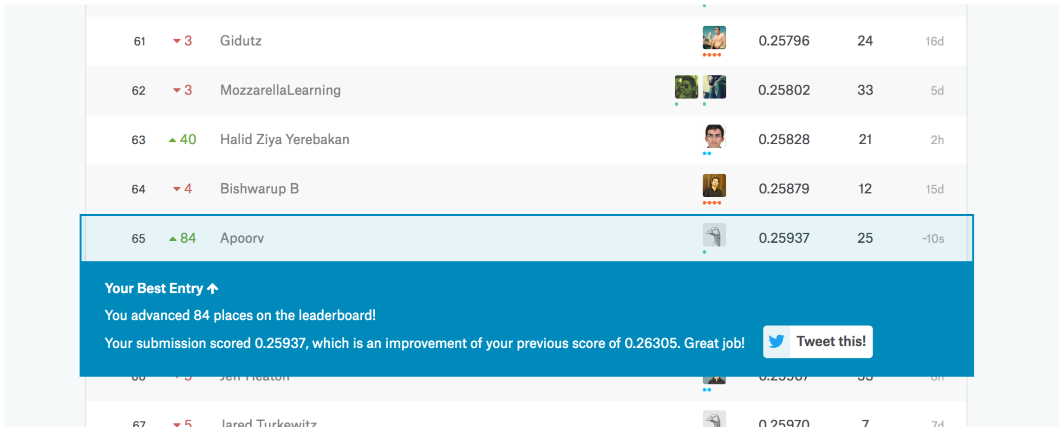
We now discuss some of the ways in which we improved our model's accuracy and log loss. We started with data augmentation and tried to incorporate Microsoft Paraphrase Corpus[1], which has

5800 pair of sentences of gold standard, along with how closely they resemble paraphrases. We also incorporate Twitter Paraphrase Corpus, which has 18000 pair of tweets along with a binary representation if they are paraphrases or not. However, due to Twitter paraphrase data being very noisy, we could not achieve any improvement on our baseline. On the contrary, Microsoft Paraphrase Corpus provided us with marginal improvements. Another technique we employed was reversing the order of question pairs and supplementing it to our dataset to make our model's parameters independent of the order of sentences. We also average the predicted probability for our test data with the orders reversed.

Adding traditional shallow learning based features like skewness and fuzzy distances to our models also provided improvement on our baseline. Another technique we used was average ensemble, where we average over our predictions of various models. We achieve this by randomizing various aspects of the model in our best model. We randomize the number of neurons in each layer between a +50/-50 range and randomize dropout probability in the range of 0.1/-0.1. We then average over the the final outputs for all variations. Another important technique we incorporated was introducing class weights into our network, because both our train and test dataset were highly unbalanced (our training dataset has 37% duplicates while our test dataset has approximately 17% duplicates ).This is carried out because models otherwise would tend to gravitate their predictions based on the inherent bias in the training dataset, and if the class ratio is not the same for the testing dataset, the model would give biased results. With class weights we can compensate the imbalance by penalizing the model differently for different classes depending on their frequency in the test dataset.

## 6 Our current ranking

We are currently at 65th rank on Kaggle. This performance was achieved by using ensemble averaging on the results obtained from our best log loss model. We hope to improve our rankings further by implementing suggestions mentioned in the "Future Work" section.



| | | | | | |
|---|---|---|---|---|---|
| 61 | ▼3 | Gidutz | 0.25796 | 24 | 16d |
| 62 | ▼3 | MozzarellaLearning | 0.25802 | 33 | 5d |
| 63 | ▲40 | Halid Ziya Yerebakan | 0.25828 | 21 | 2h |
| 64 | ▼4 | Bishwarup B | 0.25879 | 12 | 15d |
| 65 | ▲84 | Apoorv | 0.25937 | 25 | ~10s |

**Your Best Entry**

You advanced 84 places on the leaderboard!

Your submission scored 0.25937, which is an improvement of your previous score of 0.26305. Great job!

| | | | | | |
|---|---|---|---|---|---|
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| 67 | ▼5 | Jared Turkewitz | 0.25970 | 7 | 7d |

Figure 7: Our current Kaggle rank based on our best Log Loss model performance

## 7 Future Work

Some extensions on our approach include using an ensemble with 2D Convolution Networks like the approach of Kim[15]. Their study shows 2-Dimensional CNN with little hyper-parameter tuning and static word-vectors achieves excellent results on multiple benchmarks. Learning task-specific vectors by fine-tuning offers further performance gains, while allowing for minor modifications in the static embeddings and task specific vectors. They classify text-related data with the general architecture reminiscent of deep CNN based networks used in image classification . This approach could be complemented by using pre-trained CNN weights from VGG or Alexnet and fine-tuning a few top layers. Another approach involves using attention[16] on top our stacked LSTM model. The attention model allows neural networks to focus on different parts of their inputs, depending on the contribution they have on final classification. Attention has been known to improve upon existing

benchmark accuracies on various NLP tasks like translation which were solved with an LSTM based network.
In addition to this, we are also currently exploring the accuracy of BiLSTMs instead of the vanilla LSTMs that we were using in our architecture. We also plan on scraping Quora to retrieve more duplicate/non-duplicate question pairs which would augment the existing dataset and thereby improve our results further. Given that we have one month remaining for the Kaggle competition to end, we believe that these further modifications will help us achieve an even better rank eventually.

## Acknowledgements

We would like to thank Prof. Kapil Thadani and Prof. Liangliang Cao for their continuous support and guidance throughout the semester. We relied on their feedback and experience to tackle various challenges that we faced in order to improve our rank on Kaggle and validation accuracy.

## References

[1] Dolan, Bill, Chris Brockett, and Chris Quirk. "Microsoft research paraphrase corpus." Retrieved March 29 (2005): 2008.

[2] Xu, Wei, Chris Callison-Burch, and William B. Dolan. "SemEval-2015 Task 1: Paraphrase and semantic similarity in Twitter (PIT)." Proceedings of SemEval (2015).

[3] Kornél Csernai First quora dataset release: Question pairs, Jan 2017. URL `https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs`.

[4] Nikhil Dandekar Semantic Question Matching with Deep Learning. URL `https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning`.

[5] Eren Golge Duplicate Question Detection with Deep Learning on Quora Dataset URL `http://www.erogol.com/duplicate-question-detection-deep-learning/`

[6] Spacy : Industrial-Strength NLP in Python URL `https://spacy.io/`

[7] FuzzyWuzzy: Fuzzy String Matching in Python URL `https://github.com/seatgeek/fuzzywuzzy`

[8] Keras, Chollet, Francois and others, 2015, URL `Github,https://github.com/fchollet/keras`

[9] Abadi, Martín, et al. "TensorFlow: A system for large-scale machine learning." Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA. 2016.

[10] Mikolov, Tomas, et al. "word2vec." (2014).

[11] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation." EMNLP. Vol. 14. 2014.

[12] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[13] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[14] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of Machine Learning Research 15.1 (2014): 1929-1958.

[15] Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882 (2014).

[16] Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." arXiv preprint arXiv:1508.04025 (2015).