Identifying duplicated Quora questions pairs using Siamese LSTM joined with feature input

Hao Dong
University of Waterloo
200 University Avenue West Waterloo,
ON, Canada N2L 3G1
h45dong@uwaterloo.ca

Han Weng
University of Waterloo
200 University Avenue West Waterloo,
ON, Canada N2L 3G1
h5weng@uwaterloo.ca

Abstract

This document describes a mutated Siamese neural network solution to the Kaggle Quora duplicated question challenge. On a training set with 400,000 labeled question pairs, this model achieves 85% classification accuracy. The preprocessing steps are firstly introduced, including an important feature engineering work. The model features a Siamese network architecture concatenated with a feature input branch followed by stacked hidden layers before final prediction. Experiment results show that this model outperforms a simple LSTM multilayer baseline by 12%, and also has a 11% improvement by accuracy compared to conventional two-input Siamese neural network.

1 Introduction

Quora is a popular platform for people to ask questions and share their experience and insights about a topic. With over 3 million visits everyday, people ask questions about various problems they run into and therefore produces large numbers of questions that are worded similar or simply duplicated. Duplicated questions have been a headache for many Q&A-based forums(Abishek et al., 2019) as they make it less effective to find and provide information for both answer seekers and writers. Askers do not need to initiate another thread if their questions have already been answered, and the platforms usually encourage canonical questions and answers and hope similar questions to be combined or affiliated. Kaggle started a competition on this Natural Language Processing problem two years ago. The challenge attracted over 3000 groups of people around the world , and a number of other research have been conducted in the academic community, which will be reviewed in more details in Section. 2.

The datasets used in this problem contain a input table with 404k rows (`id`, `qid_1`, `qid_2`, `question1`, `question2` and a given `is_duplicate` label) and a test set with 2.35 million rows (`id`, `question1`, `question2`). The training set questions are genuine questions from Quora, but the test set contains computer-generated fake questions as an anti-cheating measure. Questions are considered duplicated as long as they express similar motivations. For example, "What is the shortest way to go from A to B" and "How to get to B from A in least time" are labeled duplicated since although the later one implies "fast" while the prior does not, they both have the motivation of asking about a direction from A to B which is fastest. However, if a question swaps A and B in the sentence, it is definitely considered different, which needs special attentions as conventional NLP approaches may see them having identical linguistic features(Addair, 2017).

2 Related works

Effort has been devoted to combining similar questions using multiple methods. For example, Quora officially uses a random forest algorithm to identify and merge similar questions. However, researches show that traditional machine learning algorithms, including support vector machine (SVM) and decision trees/random forest are outperformed by most deep learning methods(Dey et al., 2016). Most DNN models adopts a Siamese network architecture that has two input sentence vectors and combined at some stage(Homma et al., 2016), which is then connected to a conventional stacked multilayer perception machine before reaching the output layer. On top of that, some work shows that pairing this model with a convolutional neural network (CNN)

using some distance measures such as Manhattan or cosine distance also improves the performance(Bogdanova et al., 2015). There is also some work mentioning recurrent neural networks (RNN), which produced decent results on the SemEval dataset(Sanborn and Skryzalin, 2015).

We noticed that some submissions in the Kaggle competition only used feature engineering and some gradient boosting methods such as XGBoost but also achieved remarkable results. Therefore, in our model we decided to combine the popular Siamese neural network with a feature engineering branch as a makeup of sentence-level generalization.

3 Approach

3.1 Peek on the data

Before building up the solution, we first peek the data to gather some general information about how features may affect the result. This was inspired and powered by kernel code from the Kaggle competition.

Here we take length as an example feature since apart from being a factor determining the question pair is duplicated or not, we also need length information to trim sentences in word embeddings. The plot Fig. 1 shows how length difference of questions may influence the result and Fig. 2 illustrates the question length distribution. The sentence length by words is estimated by string length divided by average word length (4.6), which we estimated from sample input questions.

For other features we also check them manually by plotting, and selected the features that has most impact to include in the feature extraction part. For example, factors such as length of common substring between questions, common bigrams make significant difference on the duplicated classification. It will be discussed further in Sec. 3.3.

3.2 Preprocessing

3.2.1 Preprocessing data

Preprocessing of data is divided into two phases: one before extracting features and one after as modifying input questions may lead to bias in feature extraction. The process is shown as Fig. 3.

3.2.2 Phase I

First a data cleaning was conducted by replacing special characters with more common expression. For example, "won't" can be replaced by "will not", which suppresses false negatives on word similarity and helps reduce word occurrence matrix as well. Symbols including punctuations are removed for better generalization. From length distribution graphs from the previous section we see very few questions have length less than 2 words / 8 characters, therefore those question pairs are are discarded to reduce noise.

Lemmatization is then performed on words with length larger than 4 letters to generalize sentence semantics. We use the `WordNetLemmatizer` from `nltk.stem.wordnet` to first replace words with their noun form, and recover all nouns to verbs. To deal with rare words, we first retrive the words from GloVe embeddings as topwords set, stopwords from `ntlk.corpus` library, then replace all rarewords with placeholders according to this relationship before fed into LSTM:

$$S_{\text{rarewords}} = U \backslash (S_{\text{topwords}} \cup S_{\text{stopwords}}) \quad (1)$$

3.2.3 Tokenize

Length distribution shows that 97% of questions have length of less than 30 words. Therefore the tokenizer converts each question into a 30-dimensional vector by padding each question of both training and test dataset. The tokenized and preprocessed training and test data is used as partial input of our model. The whole vocabulary is mapped to over 80000 unique ids. This is used again when creating embedding matrix for the Embedding Layer and will be picked up again in Section. 3.4.

3.3 Feature Engineering

We put special emphasis on feature engineering in this project. For the base line of our model at the project milestone, we only used a simple DNN model with two LSTM inputs. After some case study on other competitors, we found that successful submits all included intensive feature extraction including both NLP features (sentence characteristics) and non-NLP features (kNN neighbors). For simplicity we only involve NLP features here because non-NLP features take too long time to run given the huge test set and preprocessing. After some research on the features that might be useful, we come up with the following 17 columns attached to the original data set. They can be divided into two groups by whether the comparison is strict or fuzzy. All "ratio" features include

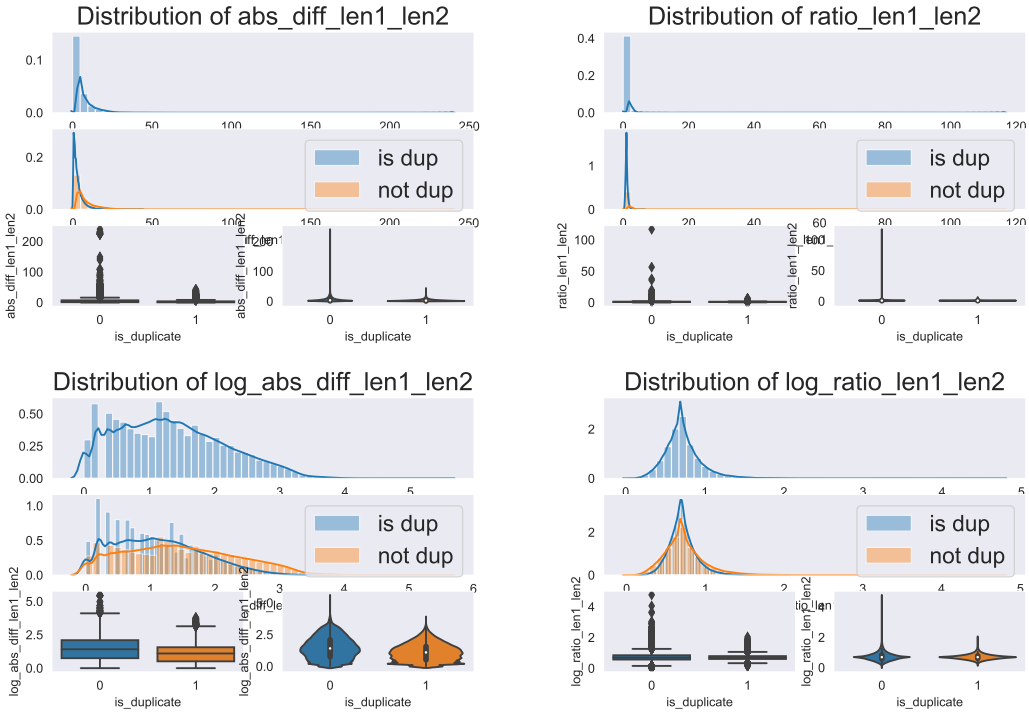

Figure 1: Question length difference and its influence on classification. Generated by kernel codes from Kaggle competition.

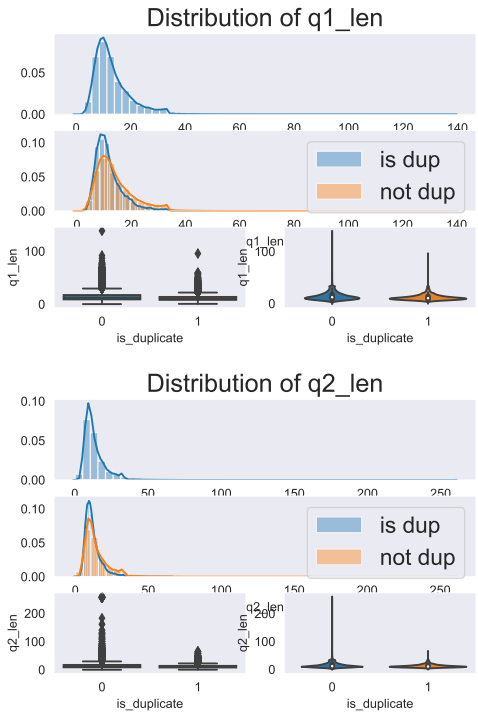

Figure 2: Question length distribution

both $N/\min(L_{q1}, L_{q2})$ and $N/\max(L_{q1}, L_{q2})$ to remove effect of absolute sentence length.

3.3.1 Strict comparison features

These include the ratio of common stopwords, unigram, bigram, nouns and substring length. We also add two columns indicating whether first and last words in questions are the same, and the length difference as well.

3.3.2 Fuzzy comparison features

With the help of `fuzzywuzzy` library, we also extracted 4 fuzzy matching features: common words ratio, common word set size ratio, common substring length ratio after sorting and partially common words ratio. This provides another layer of generalization to the solution.

The feature extraction on training and test set takes about 10 hours to run in total, and the result is saved in intermediate files which acts as the new input dataset in the following work.

3.4 Embedding

We use a pre-trained GloVe word vector to generate the embeddings. We choose the GloVe model because as a global co-occurrence matrix representation, it not only gathers global occurrence information, but also outperforms many

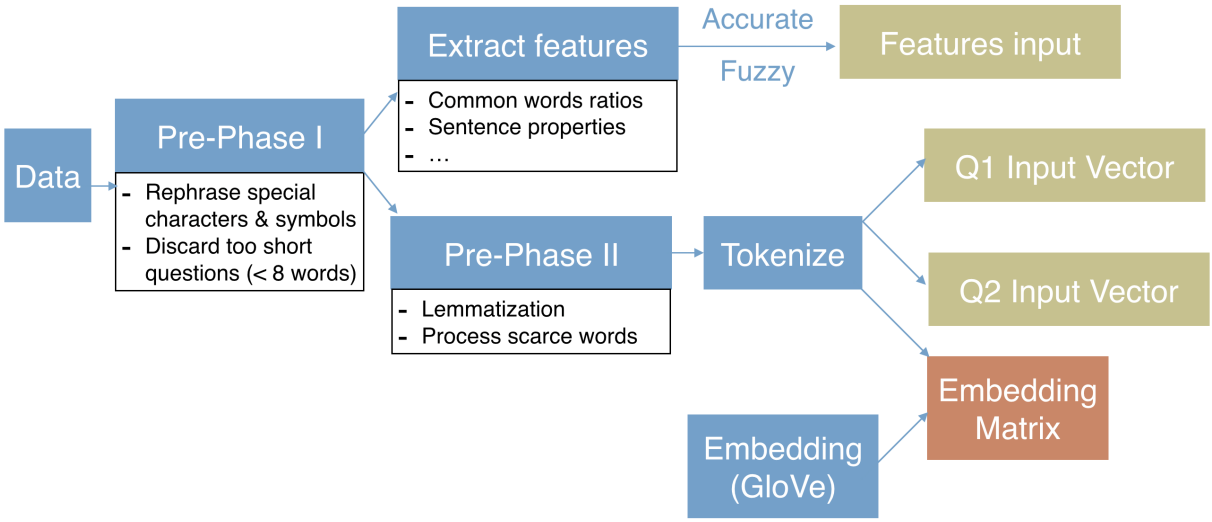

Figure 3: Pre-process workflow. Three features and one embedding matrix is obtained through this process.

local embedding methods (skip-gram, CBOW) in word analogy tasks which the later one is normally considered better at. Also GloVe results in shorter training time, and suffers from less accuracy decrease as iteration grows compared to shallow window-based methods like Word2Vec(Pennington et al., 2014). In this task, we look up the dictionary from Section. 3.2.2 and the corresponding word-embedding vectors from GloVe are assigned to the embedding matrix. If the word does not have a pre-trained embedding, then randomly initialize its embedding vector. The embedding matrix for each question is then passed through the encoding layer of our model.

3.5 Model

3.5.1 Siamese Neural Network

Siamese neural network is a class of neural network architectures that contain two or more identical sub networks which have the same configuration with the same parameters and weights, as its name suggests. For this challenge, a Siamese structure is designed specifically for question pairs in training data. To begin with, two input layers pass two question vector representations respectively to the following embedding layers. In the embedding layers, the sentence embeddings for the questions are generated and fed into the next LSTM subnetworks respectively for each question. Furthermore, a concatenation layer merges the vector representations produced by the LSTM layers into a single one. Usually distance mea-

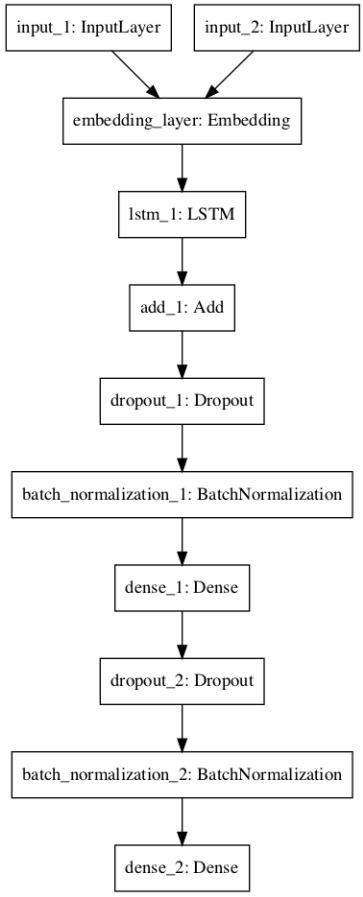

Figure 4: Simple Siamese DNN model

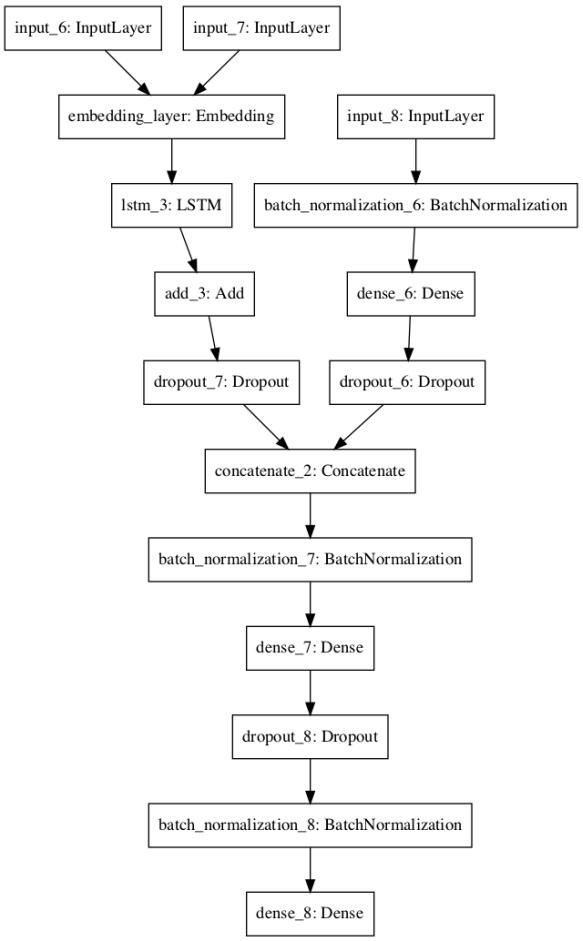

Figure 5: Siamese DNN joined with feature input

sures are involved to perform the combination, but we found a simple adding layer also produces decent result. Then, after dropout regularization against over-fitting, this intermediate output is fed into subsequent layer for further process.

3.5.2 Our Structure

Apart from the above two input layers which pass the vectors of the cleaned raw data in, some features introduced as another part of inputs are normalized to the dense layer, which uses Rectied linear unit (`ReLU`) as activation function for nonlinearity. Subsequently, the output after dropout regularization is concatenated with previous output discussed above in Section. 3.5.1 into a single vector representation. Further generalization layers are added before it goes to the final output layer with `ReLU` activation function.

Model	Loss	Accuracy
Baseline (Simple stacks)	0.586	72.6%
Simple Siamese (Fig.6a)	0.552	73.1%
Siamese+Feature(no fuzzy)	0.332	84.4%
Siamese+Feature(all)	0.323	85.0%

Table 1: Experiment Results after tuning

4 Experiment

4.1 Setup

The experiment is run on a desktop equipped with Core-i5 8400 CPU (3.9 GHz), 16 GB memory and a GTX 1070 GPU. The model is built using keras with Tensorflow-gpu backend.

4.2 Eliminate overfitting

At the milestone the model severely suffered from overfitting, with training accuracy leading validation by over 10%. We took conventional measures to lessen this, including adding dropout and normalization layers between each functional layer (LSTM, Dense) and inserting a noise layer (Gaussian Noise). Various combinations of layers and parameters were tested on these layers, which finally led to the model fixed as discussed above.

4.3 Hyperparameters

Apart from dropout parameters, some other hyperparameters are tuned on a performance-training-time balance basis. For example, we found that increasing hidden layer sizes improves the accuracy and meanwhile causes long training time. However, the performance improvement becomes very minimal after it grows over a certain number (around 150), therefore it is set at that value to achieve the balance. As for the sentence embedding size and cut length, they have already been discussed in pre-processing sections. They are set at a point where we can eliminate most noise and shorten running time, but sacrifice information integrity as little as possible.

4.4 Results

Results of three different settings are shown in Fig. 6. The Accuracy/Loss-Epoch plot shows that with all dropout and batch normalization layers added, the overfitting is well suppressed that training and validation set give a converging result. Applying NLP feature inputs on top of just LSTM results in 12% increase in accuracy and fuzzy features give a rise of another 1%. Also, the introduc-

tion of addtional features make validation results fluctuates less. The best model uses a Siamese network with LSTM layers and a branch introducing feature engineering results (with fuzzy features).

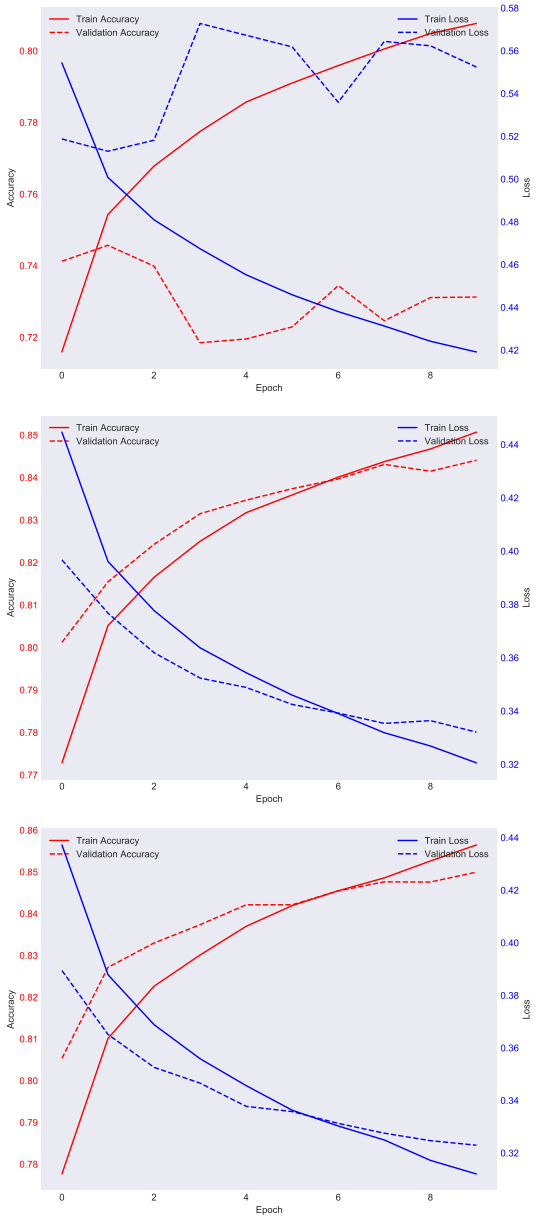

Figure 6: Fitting result. Top: model with no feature input. Middle: model with only strict matching features. Bottom: model with fuzzy features.

It achieves 85% accuracy and 0.32 loss on the validation set. The submitted test result achieves 0.31 loss on Kaggle, which now beats 70% submissions. However, we notice that most leading submissions now are practice replicas of some shared prize-winning codes since this competition was 2 years ago. According to a report from that time (from a group of STAT student in Waterloo, saying 0.2844 ranked top 8%[1]), our result should have ranked about top 10% in this competition.

5 Conclusion

In this document, we introduced a slightly mutated Siamese neural network with LSTM input layers and GloVe embeddings to tackle the problem of recognizing duplicated Quora questions. Apart from conventional preprocessing and model building, we put much effort in performing feature engineering on a side branch, which turned out significantly increased the performance model. We conclude the reason as that although the embedding and sentence vectors (`input_1` and `input_2`) express the vectorized meaning based on per-trained corpus, they do not carry enough contextual information of the sentence similarity, which is made up by manually controlled features fed into the network from `input_3` branch. The single model produces a result of (85% accuracy/0.32 loss) and ranks 10% (2017, competition time parity). If time allows, we think training multiple models and averaging the predictions may leads to a better result.

References

K Abishek, Basuthkar Rajaram Hariharan, and C Valliyammai. 2019. An enhanced deep learning model for duplicate question pairs recognition. In *Soft Computing in Data Analytics*, pages 769–777. Springer.

Travis Addair. 2017. Duplicate question pair detection with deep learning. *Stanf. Univ. J.*

Dasha Bogdanova, Cicero dos Santos, Luciano Barbosa, and Bianca Zadrozny. 2015. Detecting semantically equivalent questions in online user forums. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 123–131.

Kuntal Dey, Ritvik Shrivastava, and Saroj Kaushik. 2016. A paraphrase and semantic similarity detection system for user generated short-text content on microblogs. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2880–2890.

Yushi Homma, Stuart Sy, and Christopher Yeh. 2016. Detecting duplicate questions with deep learning. In *Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS 2016). IEEE*, pages 1–8.

[1] http://static.hongbozhang.me/doc/STAT_441_Report.pdf

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Adrian Sanborn and Jacek Skryzalin. 2015. Deep learning for semantic similarity. *CS224d: Deep Learning for Natural Language Processing Stanford, CA, USA: Stanford University*.