Machine Learning Engineer Nanodegree Capstone Project

Question Similarity Predictions using Siamese LSTM Networks

Akila Jeeson Daniel

October 21th, 2017

I. Definition

Project Overview

Natural Language Processing (NLP) is field in computer science/artificial intelligence which deals with interaction of computers with human natural languages. There are different topics of research such as language understanding, replication, summarizing, translations. One of the fundamental problems to solve is how two sentences are similar. This comes under the topic of Paraphrase Matching in NLP. We are trying to quantify the similarity in the intent of two sentences as understood by a human reviewer. This is a current research problem with many solutions being proposed on the academia standard dataset as Microsoft Research Paraphrase Corpus (https://aclweb.org/aclwiki/Paraphrase_Identification_(State_of_the_art)).

The problem we are trying to solve is to predict - 'How similar are two questions?'. This specific problem was addressed as a 2016 Kaggle competition organised by Quora. The dataset we are going to use for our analysis is the Quora Question Pairs dataset (https://www.kaggle.com/quora/question-pairs-dataset).

Problem Statement

The problem statement we are trying to solve in this project is - 'How similar are two questions?'. As a baseline approach, we can convert sentences to vectors, use cosine distance between vectors to measure sentence similarity and then train a basic machine learning model such as logistic regression to predict duplicates.

We can improve on our benchmark model by using advanced word to vector conversion models such as Word2Vec or GloVe.6B and then use an advanced recurrent neural networks such as LSTM or GRU to better predict if the questions are duplicate or not within each pair. This project is inspired by - Bogdanova 2015 (https://aclweb.org/anthology/K15-1013), Addair 2016 (https://web.stanford.edu/class/cs224n/reports/2759336.pdf), Homma et al. 2017 (https://web.stanford.edu/class/cs224n/reports/2748045.pdf), Kaggle kernels by Lystdo (https://www.kaggle.com/lystdo/lstm-with-word2vec-embeddings).

Metrics

To evaluate the performance of our models, we use accuracy and binary log-loss.

For a sample of N objects (question pairs) which can be classified into positive (is_duplicate = 1) or negative (is_duplicate = 0), binary log loss is defined as

$$-\frac{1}{N}\sum_{i=1}^{N}[y_i log\, p_i + (1 - y_i)\, log\, (1 - p_i)]$$

where y_i is the correct classification for sample i, and p_{ij} is the model probability of assigning positive class to sample i.

Accuracy can be defined as -

TP - True Positive = Number of objects in Positive class predicted by the model to be in the positive class.

FP - False Positive = Number of objects in Negative class predicted by the model to be in the positive class.

TN - True Negative = Number of objects in Negative class predicted by the model to be in the negative class.

FN - False Negative = Number of objects in Positive class predicted by the model to be in the negative class.

Accuracy = TP + TN/ (TP + TN + FP + FN)

For our problem these metrics makes sense because we want a model which predicts the appropriate classes as accurately as it can while reducing both false positives and false negatives. Log Loss quantifies the accuracy of a classifier by penalising false classifications. Minimizing log loss will help us achieve our optimal high accuracy.

II. Analysis

Data Exploration

The dataset we are going to use for our analysis is the Quora Question Pairs dataset (https://www.kaggle.com/quora/question-pairs-dataset). The dataset consists of 404350 question pairs which as labeled as duplicates (1) or not (0) which 149306 are in the 'is_duplicate = 1' class. Examples of the data are shown below -

Id	0
qid1	1
qid2	2
question1	What is the step by step guide to invest in share market in india?
question2	What is the step by step guide to invest in share market?
is_duplicate	0

id	7
qid1	15
qid2	16
question1	How can I be a good geologist?
question2	What should I do to be a great geologist?
is_duplicate	1

There are 789801 different questions (as represented by qid1 and qid2) in the dataset. There are only 2 questions which have null values. The mean number of words in each question about about 60. If we remove the stopwords (using NLTK 'English' stopwords list), there are about 37 words in each question on average. There are a total of 85335 distinct words in the question list.

Algorithms and Techniques

For this project, I use Python 2.7 with libraries such as Pandas, scikit-learn, Keras with Tensor-flow backend.

We split the questions to word lists and convert wordlists to sentence vectors. For our benchmark model, we use these word lists to c compute the term frequency-inverse document frequency (TFIDF) as the weights for words in the question vector. TFIDF computes how important a word is to a question.
TF(w) = Number of times a word w appears in a question / Total number of words in the question.
IDF(w) = ln(Total number of question / Number of question with word w)

TFIDF(w) = TF(w) * IDF(w).

With the word vector for each question, we compute the cosine similarity between the question pairs. Cosine similarity is the cosine of the angle between the two vectors. For vectors A and B of magnitude |A| and |B|,
cosine similarity is defined as - Dot_product(A,B)/(|A||B|).
Finally we use a logistic regression model to classify the cosine similarity of vectors into duplicates and not duplicates classes. Logistic regression fits a sigmoid function (s(t) = 1/(1+e^-t)) to the data to determine above what threshold, the sigmoid should predict duplicates class.

To improve on our benchmark model tfidf word representations, we use pretrained word representation models such as Word2Vec [https://code.google.com/archive/p/word2vec/] and GloVe.6B [https://nlp.stanford.edu/projects/glove/]. The Word2Vec word vector representations (GoogleNews-vectors-negative300.bin.gz) were created by training word2vec's "skip-gram and CBOW models" on Google News corpus (3 billion running words) to produce 300-dimension English word vectors. The GloVe.6B word vector representations [glove.6B.zip] was trained on word-word co-occurrences statistics from 6Billion tokens [400K vocab] of Wikipedia 2014 + Gigaword 5 datasets using the GloVe model [https://nlp.stanford.edu/pubs/glove.pdf]. We tried both these word embeddings and found that a larger number of the words in our Quora vocabulary is present in the GloVe embedding vocabulary compared to the Word2Vec vocabulary. For 85335 words from 404350 question pairs, Glove give 24517 null embedding compared to Word2Vec's 37199 null embeddings. Therefore, we chose GloVe word vector representations for our final model.

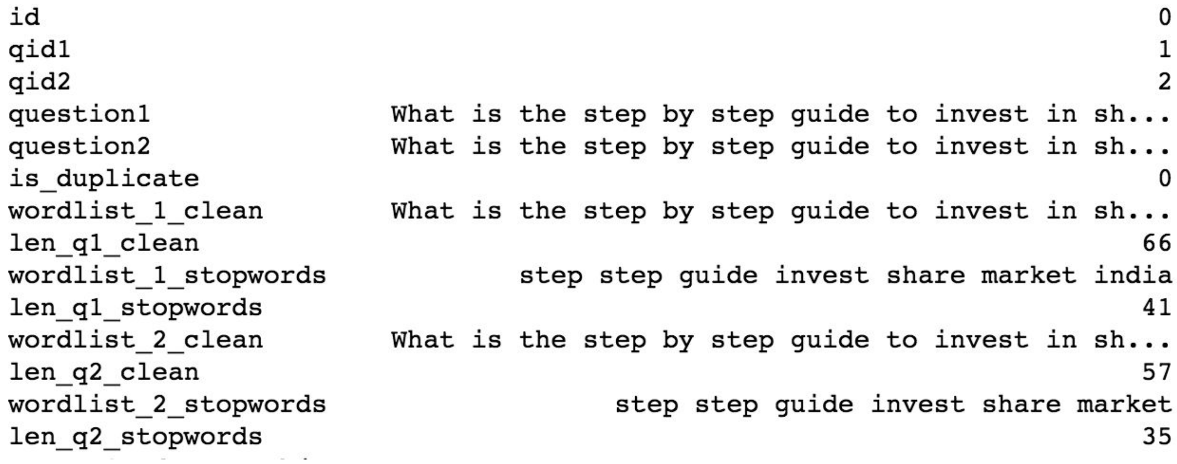
For our machine learning side of the model, we tried two types of Recurrent Neural Networks (RNN) - Long Short Term Memory networks (LSTM, http://people.idsia.ch/~juergen/rnn.html) and Gated Recurrent Units (GRU, Cho et al. 2014 https://arxiv.org/abs/1406.1078). RNNs are ideal for NLP tasks involving sequence of words as they can remember information regarding previous inputs. All RNNs have a chain of repeating neural networks [http://colah.github.io/posts/2015-08-Understanding-LSTMs/]. LSTM passes on information regarding previous inputs by passing on to the 'hidden' state and the 'cell state' to the next neural network in the chain which then receives the new input. A GRU is similar to an LSTM in the way that it passes on the hidden state to the next neural network in the chain but it does not have a 'cell state'. Thus GRU has fewer parameters than LSTM and is thus faster to compute. Later, we find that in our case, LSTM works better than GRU for the same number of parameters.

We also use a siamese architecture for our RNNs and non-trainable embedding layers which has been shown to work well with these tasks [e.g. https://web.stanford.edu/class/cs224n/reports/2748045.pdf, https://www.kaggle.com/lystdo/lstm-with-word2vec-embeddings). In siamese network model, we train both the questions in each pair using the same RNN/Embedding model which shares parameters between the trainings.

Data Cleaning

Our primary data cleaning process is as follows -
1. We removed punctuations, word shortenings (e.g. “ ‘re ”,” ’s ”,” ’ll ”,” ’d “) from the question text. This is saved as the ‘*_clean’ text where ‘*’ is for words from question1 or question2.
2. Next, we remove stopwords (using NLTK’s ‘English’ stopwords corpus) from the ‘*_clean’ text and save question text as ‘*_stopwords’.

An example of the cleaned dataset is shown below -



```
id                                                                     0
qid1                                                                   1
qid2                                                                   2
question1              What is the step by step guide to invest in sh...
question2              What is the step by step guide to invest in sh...
is_duplicate                                                           0
wordlist_1_clean       What is the step by step guide to invest in sh...
len_q1_clean                                                          66
wordlist_1_stopwords           step step guide invest share market india
len_q1_stopwords                                                      41
wordlist_2_clean       What is the step by step guide to invest in sh...
len_q2_clean                                                          57
wordlist_2_stopwords                 step step guide invest share market
len_q2_stopwords                                                      35
```

The labels for question 1/2 are - ‘wordlist_1/2_clean’ for cleaned wordlist, ‘len_q1/2_clean’ for the number of words in the cleaned word list, ‘wordlist_1/2_stopwords’ for stopwords removed clean wordlist and ‘len_q1/2_stopwords’ for number of words after removing stopwords.

Figure 1 shows the probability density function (y-axis) of the number of words in the questions in lists question1 and question2 (x-axis) with stop words (‘len_q*_clean’) and without stopwords (‘len_q*_stopwords’). They are both right-tailed distributions with the max word count for questions including stopwords is 1171 with a median value of 52. Without stopwords, the maximum word count is 673 with a median of 32. We can see that most of the sentences have a word count less than 150, but as a conservative limit we pad our sequences to length of 300 words for batch processing our our RNN models.

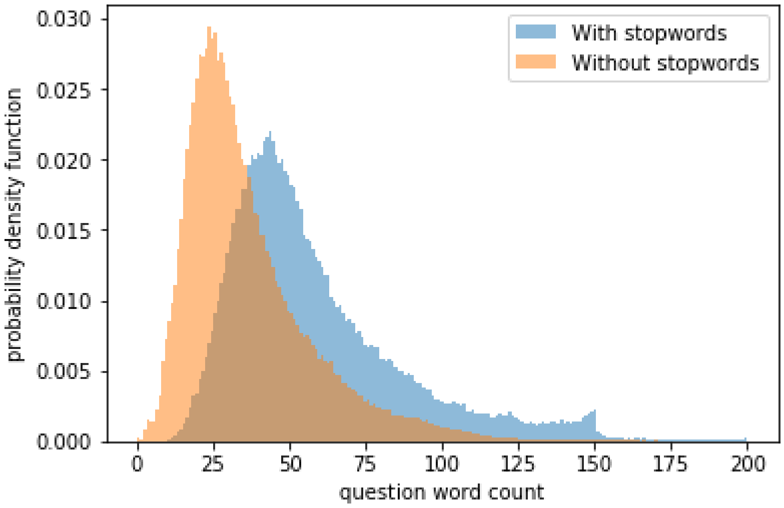


Figure 1. The probability density function of words in cleaned questions with and without stopwords included.

Benchmark

Our benchmark model was built as follows -

!. The first step is to clean the data.

2. Next step involves converting question texts to vector representations of sentences. We use term frequency inverse document frequency (TFIDF) weighted word vectors to build our sentence vectors. Most of our text processing was done using Keras.preprocessing.text.tokenizer. The tokenizer used all the questions text to produce word lists, word frequency and TFIDF. Then we used the tokenizer to convert each question text to word vectors with the word weights given by TFIDF. We tested our models with ‘*_clean’ and ‘*_stopwords’ and found that stopword removal takes away some of our accuracy. Therefore, for our final benchmark model, we use ‘*_clean’ text.

3. Next, to combine the two sentence vectors to find the cosine similarity between question pairs. We used pairwise.cosine_similarity from sklearn.metrics.

4. Finally we use logistic regression to determine at what threshold of cosine similarity distance, does the log-loss of predictions, reach a minimum value. We use sklearn.linear_model.LogisticRegression with fine tuning of the regularizarion parameter C.

Due to the training costs, we perform our modelling on a sub-dataset of 50000 question pairs. We expect the results to hold for even larger datasets. We split 30% of our sub-dataset for testing. Our best fit model fit has an accuracy of 0.6652 and log_loss of 0.6008 for a C = 0.1 for l2 penalty with ‘lbfgs’ solver. We find that the decision threshold given by logistic regression is between 0.52 and 0.53 for cosine similarity.

Therefore, with our RNN models, we aim to reduce our log-loss from 0.6008 and achieve an accuracy higher than 0.6652 compared to our benchmark model.

III. Methodology

Data Preprocessing

Before applying machine learning to our data, we need to clean the data and convert it to input features for our machine learning models. We aim to convert our questions into word vectors in different methods such as TFIDF, Word2Vec or GloVe.6B embeddings.

The data cleaning process is the same for all the models and is explained in the Data Cleaning section above. For the benchmark model, we used the TFIDF vector representation for sentences. For our RNN models, we used advanced word vector representations - Word2Vec or GloVe.6B. For both these word vector representations, we used the '*_clean' wordlists to build the embedding matrix where for each word in the list, we find the corresponding word in the embedding vocabulary and add it to the embedding matrix. The embedding matrix has the dimension of total number of words in our wordlist times length of the word vector representations which is set to 300 by our embedding models. For the two null values in our '*_clean' data, we replaced them with ''. For 85335 words from 404350 question pairs, Glove give 24517 null embedding compared to Word2Vec's 37199 null embeddings. Therefore, we chose GloVe word vector representations for our final model.

Implementation

For our deep learning models, we use a siamese RNN architecture with GloVe.6B embeddings. We use Keras module with a TensorFlow backend and run the models on NVIDIA Tesla K80 GPU as supported by Crestle.com.

Figure 2 shows the RNN network architecture we implemented. The questions are split into tokens by the tokenizer and padded to a uniform sequence length of 300 for batch processing. These question sequences are passed through an embedding layer with GloVe.6B based embedding matrix for which the parameters are non-trainable. Since the training process is computationally intensive, we decided to use the pre-trained weights. The embedding layer outputs are then passed on a siamese LSTM/GRU layer. The two questions in each pair are trained separately using the same LSTM/GRU layer which shares the weights between trainings.

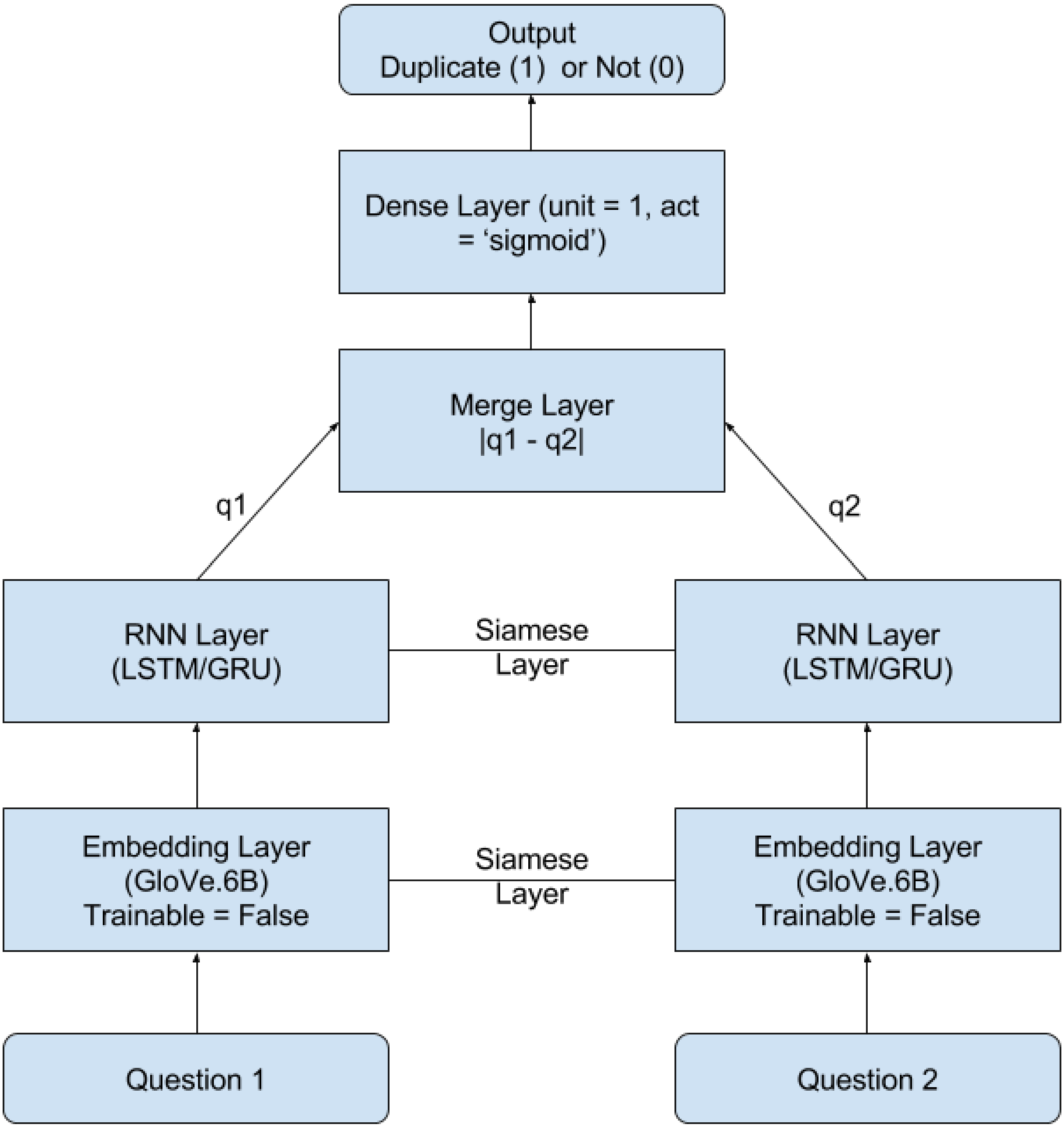


Figure 2. Our RNN Network Architecture.

The LSTM layer had 70 units and the GRU layer had 90 units both amounting to about 100,000 parameters to train, A value of 0.2 was used for both the dropout_rate and the reccurent_dropout rate. The outputs of the LSTM/GRU layer, q1 and q2, are then merged using a merge layer using the L1 distance formula abs(q1 - q2) [https://sorenbouma.github.io/blog/oneshot/]. The merge layer output is then passed on to a dense layer with a single unit and 'sigmoid' activation to compute the output of whether the pairs are Duplicate(1) or Not (0). The dense layer had 71 parameters for the LSTM model and 91 parameters for the GRU model to train.

To optimize our models, we used a ‘Nadam’ optimizer which is expected to do better than ‘Adam’ and ‘RMSProp’ (http://cs229.stanford.edu/proj2015/054_report.pdf) for similar tasks. Nadam is a gradient descent optimization algorithm with adaptive learning rates (like RMSProp, Adam. http://ruder.io/optimizing-gradient-descent/index.html#nadam) and Nestov accelerated gradient momentum.

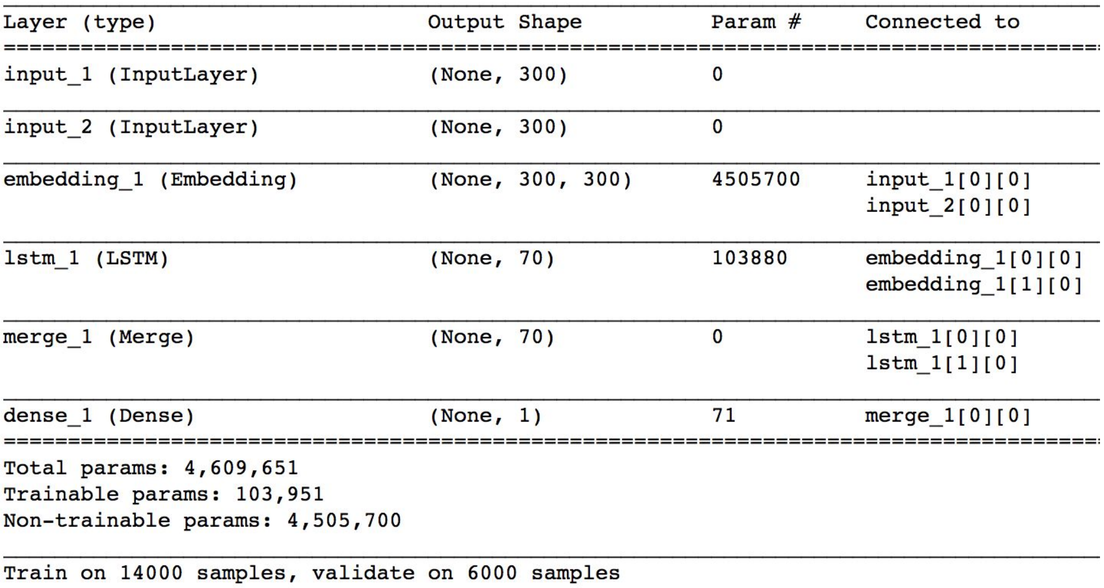
Since we have binary outputs, we use ‘binary_crossentropy’ loss function (equivalent to binary log loss) and accuracy to obtain the optimized model. The best fit model reaches a minimum loss value on our validation dataset. The keras model fitting is run for upto 200 epochs with an early stopping mechanism when the validation loss function ‘val_loss’ reaches a minimum value and starts increasing again for 3 more epochs (patience = 3). Most training stopped in less than 20 epochs.

In our original proposal, we expected to use Accuracy and f-score as our metrics but since Keras does not have f-score, we switched to using log-loss for our project which is similar to f-score in accounting for false positives and false negatives in our models.

One of the main challenge we faced during our implementation was data size and RAM size. For example, when we tried to store the TFIDF vectors for all the questions in our dataset (each question vector with 33298 floats), the machine ran out of RAM and crashed before we could compute cosine similarities. To solve this, we saved the sequences of each of the question list to intermediate CSV files which were then read simultaneously iteratively to compute cosine similarity for each question pair. Another interesting implementation challenge was to speed up the machine learning training process. We found that using a GPU speed up the training calculations by factor of 5-10.

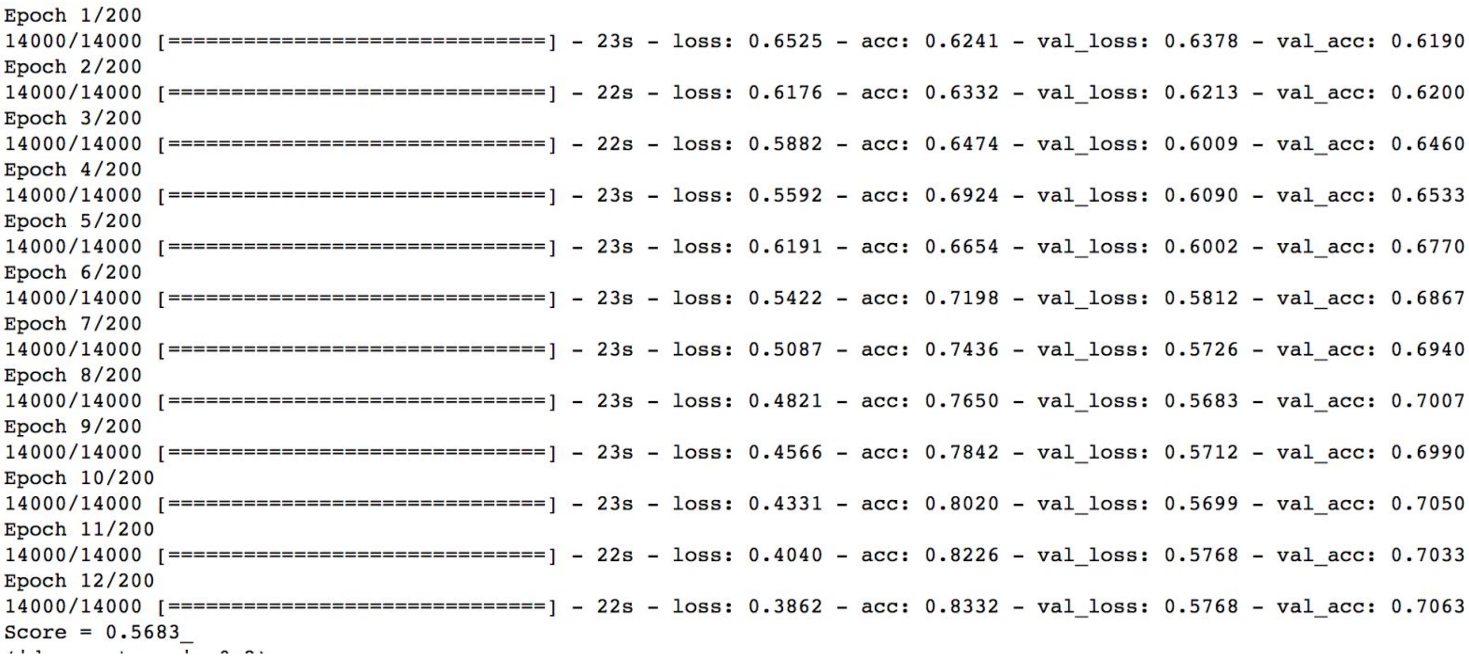
Refinement

Similar to our benchmark model, our dataset for our RNN models is a 50,000 question pairs subset of our full dataset of 400K pairs. We split 30% of our 50K dataset for validation and rest for training. We also extract a separate test dataset from our full dataset (400K pairs) with the same size as our validation set for independent final testing. Our best fit parameter results are explained in the next section. In this section, we explain our parameter testing and model fine tuning. Since the 50K dataset runs take a few hours, to do parameter testing, we used a smaller dataset of 10,000 question pairs with 30% split to validation set and 70% for training set. A trick we use to remove symmetry issues in our model and increase our training size is to flip each question pair (question 2 as q1 and vice versa). This gives us a training size of 14000 question pairs and 6000 validation pairs. For our baseline RNN parameter testing model, we have an LSTM model with 100K parameters (70 units).



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 300)           0
____________________________________________________________________________________________________
input_2 (InputLayer)             (None, 300)           0
____________________________________________________________________________________________________
embedding_1 (Embedding)          (None, 300, 300)      4505700     input_1[0][0]
                                                                   input_2[0][0]
____________________________________________________________________________________________________
lstm_1 (LSTM)                    (None, 70)            103880      embedding_1[0][0]
                                                                   embedding_1[1][0]
____________________________________________________________________________________________________
merge_1 (Merge)                  (None, 70)            0           lstm_1[0][0]
                                                                   lstm_1[1][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 1)             71          merge_1[0][0]
====================================================================================================
Total params: 4,609,651
Trainable params: 103,951
Non-trainable params: 4,505,700
____________________________________________________________________________________________________
Train on 14000 samples, validate on 6000 samples
```

Figure 3. Screenshot of our baseline LSTM parameter testing model.



```
Epoch 1/200
14000/14000 [==============================] - 23s - loss: 0.6525 - acc: 0.6241 - val_loss: 0.6378 - val_acc: 0.6190
Epoch 2/200
14000/14000 [==============================] - 22s - loss: 0.6176 - acc: 0.6332 - val_loss: 0.6213 - val_acc: 0.6200
Epoch 3/200
14000/14000 [==============================] - 22s - loss: 0.5882 - acc: 0.6474 - val_loss: 0.6009 - val_acc: 0.6460
Epoch 4/200
14000/14000 [==============================] - 23s - loss: 0.5592 - acc: 0.6924 - val_loss: 0.6090 - val_acc: 0.6533
Epoch 5/200
14000/14000 [==============================] - 23s - loss: 0.6191 - acc: 0.6654 - val_loss: 0.6002 - val_acc: 0.6770
Epoch 6/200
14000/14000 [==============================] - 23s - loss: 0.5422 - acc: 0.7198 - val_loss: 0.5812 - val_acc: 0.6867
Epoch 7/200
14000/14000 [==============================] - 23s - loss: 0.5087 - acc: 0.7436 - val_loss: 0.5726 - val_acc: 0.6940
Epoch 8/200
14000/14000 [==============================] - 23s - loss: 0.4821 - acc: 0.7650 - val_loss: 0.5683 - val_acc: 0.7007
Epoch 9/200
14000/14000 [==============================] - 23s - loss: 0.4566 - acc: 0.7842 - val_loss: 0.5712 - val_acc: 0.6990
Epoch 10/200
14000/14000 [==============================] - 23s - loss: 0.4331 - acc: 0.8020 - val_loss: 0.5699 - val_acc: 0.7050
Epoch 11/200
14000/14000 [==============================] - 22s - loss: 0.4040 - acc: 0.8226 - val_loss: 0.5768 - val_acc: 0.7033
Epoch 12/200
14000/14000 [==============================] - 22s - loss: 0.3862 - acc: 0.8332 - val_loss: 0.5768 - val_acc: 0.7063
Score = 0.5683_
```

Figure 4. Screenshot of the training epochs and loss and accuracy for training and validation datasets in each epoch. The score shows the minimum validation log loss value ‘val_loss’. Epoch 8 has score = val_loss = 0.5683 has the optimal solution. The training loss, loss = 0.4821, training accuracy, acc = 0.7650, validation loss, val_loss = 0.5683 and validation accuracy, val_acc = 0.7007. Since validation dataset is separate from the training dataset, we can compare these values to our TFIDF benchmark model loss = 0.6008 and accuracy values = 0.6652. We can see that our

RNN baseline trained on 14000 pairs already provides a better model that our TFIDF baseline model trained on 35000 pairs. Our full model trained on 35000 (*2) pairs provides a significant improvement on the TFIDF baseline as will be shown in the results section.

LSTM vs GRU - For the same model as above but with a GRU layer of 90 units, instead of the LSTM layer, gives these metrics for the optimal epoch - loss: 0.5135 - acc: 0.7285 - val_loss: 0.5999 - val_acc: 0.6730. The val_loss is higher than the LSTM model value of 0.5683. Thus, we chose an LSTM layer instead of a GRU layer for the final RNN model.

LSTM trainable parameters [approximately 50K,100K, 250K, 400K] -

1. 54560 parameters with 40 LSTM units - loss: 0.4834 - acc: 0.7651 - val_loss: 0.5743 - val_acc: 0.6837
2. 103880 parameters with 70 LSTM units - loss: 0.4821 - acc: 0.7650 - val_loss: 0.5683 - val_acc: 0.7007
3. 270600 parameters with 150 LSTM units - loss: 0.4836 - acc: 0.7682 - val_loss: 0.5699 - val_acc: 0.6967
4. 400800 parameters with 200 LSTM units - loss: 0.4422 - acc: 0.7957 - val_loss: 0.5690 - val_acc: 0.7007

It seems that we need atleast 100K parameters for our best fit model Therefore, for our final model, we chose an LSTM with 70 units.

GloVe.6B vs Word2Vec embeddings -
GloVe.6B embeddings with about 29% null embedding for our 85K word tokens - loss: 0.4821 - acc: 0.7650 - val_loss: 0.5683 - val_acc: 0.7007
Word2Vec embeddings with about 43% null embedding for our 85K word tokens - loss: 0.4890 - acc: 0.7574 - val_loss: 0.5802 - val_acc: 0.6880
Word2Vec model val_loss is higher than the GloVe.6B model, keeping all other parameters the same. Thus, we can see GloVe.6B embedding gives us a better model than Word2Vec embeddings.

Dropout rate (same value for recurrent drop out rate) = 0.1 or 0.2 or 0.4 -
Dropout rate 0.1 - loss: 0.4756 - acc: 0.7733 - val_loss: 0.5765 - val_acc: 0.6913
Dropout rate 0.2 - loss: 0.4821 - acc: 0.7650 - val_loss: 0.5683 - val_acc: 0.7007
Dropout rate 0.4 - loss: 0.4871 - acc: 0.7619 - val_loss: 0.5893 - val_acc: 0.6930
Dropout rate of 0.2 has the lowest val-loss among the three dropout values. Therefore, we chose 0.2 for our final model.

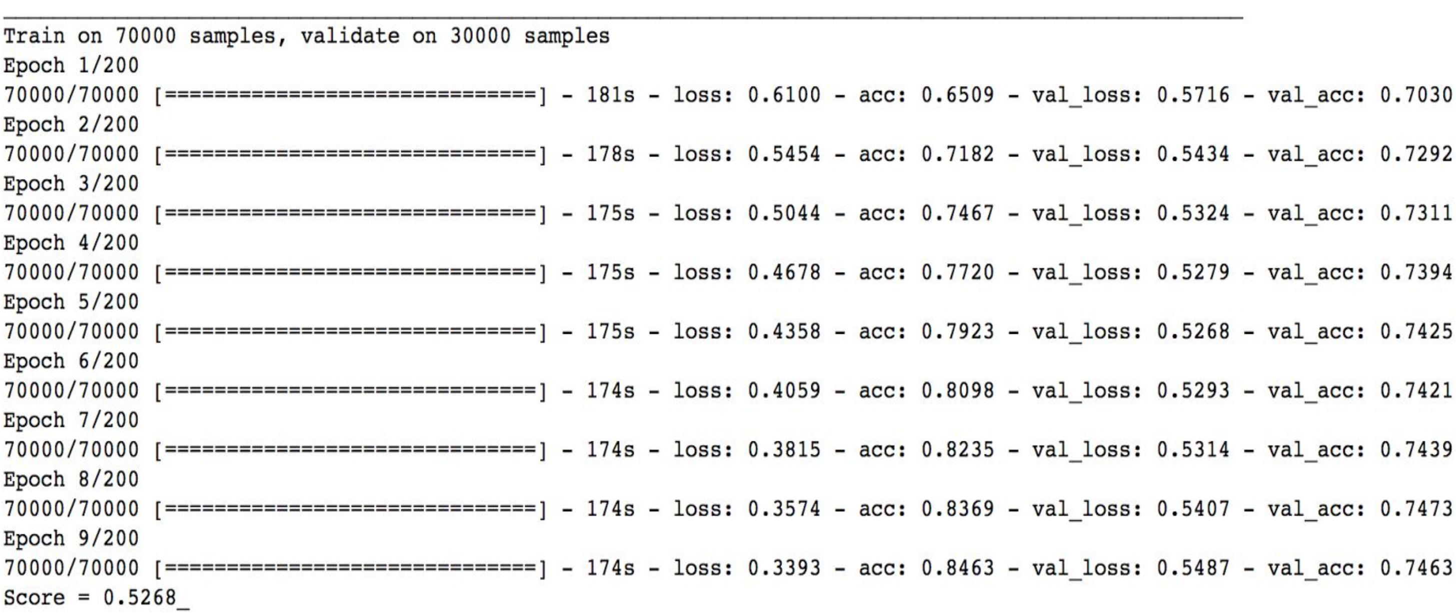
For our final model, we have these parameters - LSTM layer with 70 units (100K parameters), GloVe.6B word vector embeddings, dropout value = 0.2.

IV. Results

Model Evaluation and Validation

For the final model, we increase the dataset size from 10000 to 50000 pairs - to 35000 (*2) training pairs, 15000 (*2) validation pairs. We also select a separate set of 15000 pairs as an independent test set for our final model. The final model parameters are - LSTM layer with 70 units (100K parameters), GloVe.6B word vector embeddings, dropout value = 0.2.

For our final model, the best fit metrics are - loss: 0.4358 - acc: 0.7923 - val_loss: 0.5268 - val_acc: 0.7425. Comparing these to the same model architecture trained on 10,000 pairs [loss: 0.4821 - acc: 0.7650 - val_loss: 0.5683 - val_acc: 0.7007], we can see that increasing the training dataset size has significantly improved our val_acc from 0.7007 to 0.7425. Similarly, the val_loss has dropped from 0.0.5683 to 0.5268. On our independent test set with 15000 pairs, the predictions have an accuracy of 0.7388 and log loss of 0.5310 which are close to our validation metrics thus proving the strength of our model. Running the same model with different input data provides similar values for metrics thus showing that our model is robust to small variations in inputs.



```
Train on 70000 samples, validate on 30000 samples
Epoch 1/200
70000/70000 [==============================] - 181s - loss: 0.6100 - acc: 0.6509 - val_loss: 0.5716 - val_acc: 0.7030
Epoch 2/200
70000/70000 [==============================] - 178s - loss: 0.5454 - acc: 0.7182 - val_loss: 0.5434 - val_acc: 0.7292
Epoch 3/200
70000/70000 [==============================] - 175s - loss: 0.5044 - acc: 0.7467 - val_loss: 0.5324 - val_acc: 0.7311
Epoch 4/200
70000/70000 [==============================] - 175s - loss: 0.4678 - acc: 0.7720 - val_loss: 0.5279 - val_acc: 0.7394
Epoch 5/200
70000/70000 [==============================] - 175s - loss: 0.4358 - acc: 0.7923 - val_loss: 0.5268 - val_acc: 0.7425
Epoch 6/200
70000/70000 [==============================] - 174s - loss: 0.4059 - acc: 0.8098 - val_loss: 0.5293 - val_acc: 0.7421
Epoch 7/200
70000/70000 [==============================] - 174s - loss: 0.3815 - acc: 0.8235 - val_loss: 0.5314 - val_acc: 0.7439
Epoch 8/200
70000/70000 [==============================] - 174s - loss: 0.3574 - acc: 0.8369 - val_loss: 0.5407 - val_acc: 0.7473
Epoch 9/200
70000/70000 [==============================] - 174s - loss: 0.3393 - acc: 0.8463 - val_loss: 0.5487 - val_acc: 0.7463
Score = 0.5268_
```

Figure 5. Results for the run of final model.

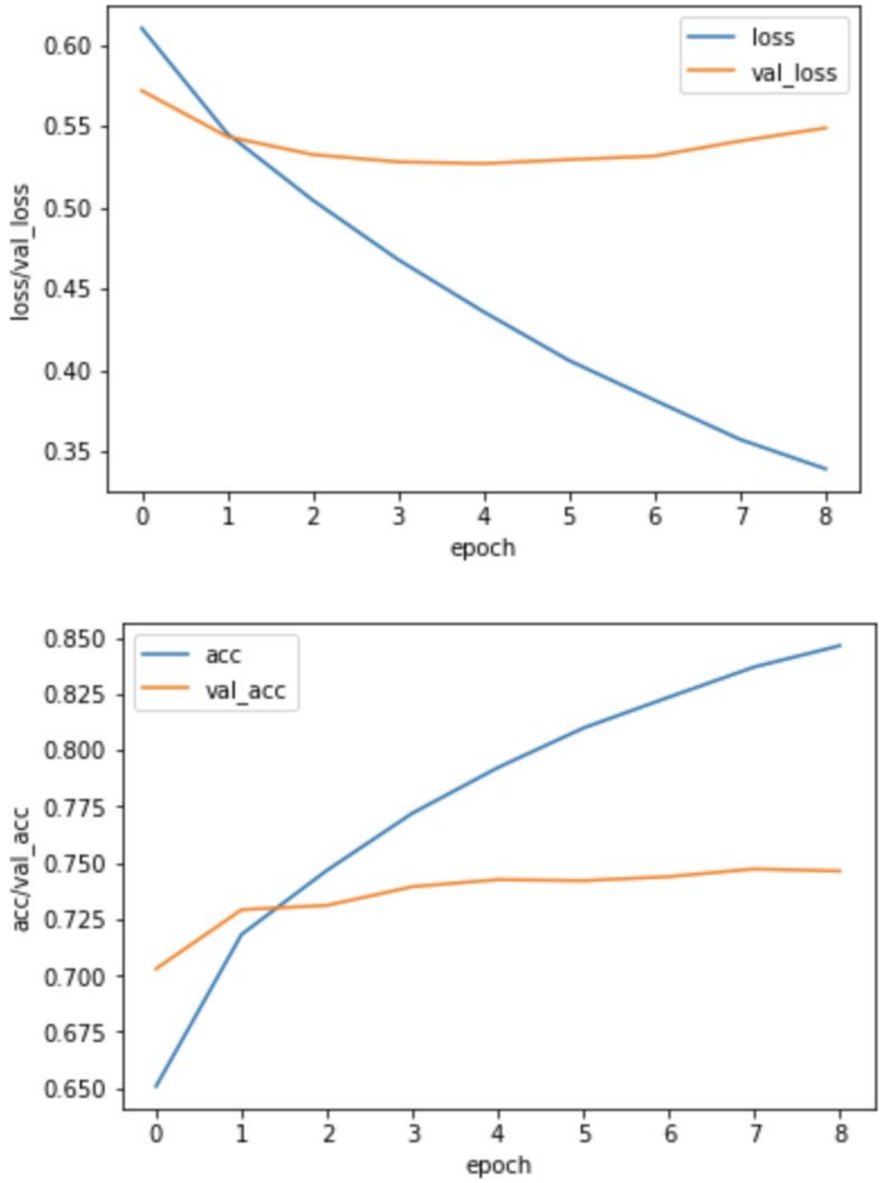


Figure 6. The evolution of log - loss (top) and accuracy (bottom) for training and test sets with each epoch. For log-loss figure, the x-axis shown the number of epochs and the y-axis is the log loss value. The training loss seems to decrease with epochs due to overfitting while the validation loss flattens below 0.55 around epoch 4-5 before increasing again. For the accuracy figure, the x-axis is the epoch and the y-axis is the accuracy value. Similar to the log-loss case, at later epochs the training accuracy keeps increasing while the validation accuracy flattens to values above 0.725. Therefore, the model around the epoch 5 with the minimum val_loss is chosen as the best model while avoid overfitting.

Justification

Comparing our best fit model to our original TFIDF benchmark metrics -
TFIDF benchmark - accuracy of 0.6652 and log_loss of 0.6008.
Best fit RNN model - test accuracy of 0.7388 and log loss of 0.5310.

Our best fit RNN model has an accuracy of 0.7388 which is greater than 0.6652 and a log loss of 0.5310 which is less than 0.6008. Thus our best fit LSTM model is a better solution than the TF-IDF model.

To test how sensitive is our model to training inputs and the range in accuracy and log_loss values, we repeated the experiment with a separate 50,000 pair dataset, with no overlapping pairs [between question pair ‘id’ of 0 and 355000] and obtained these results. For test accuracy, we have [0.7486,0.7409,0.7489,0.7503,0.7388] which has a mean of 0.7455 and standard deviation of 0.0045. For test log-loss, we have [0.5075,0.5283,0.5225,0.5144,0.5310] which has a mean of 0.5207 and a standard deviation of 0.0087. Therefore, our TFIDF benchmark accuracy of 0.6652 is greater than 17 standard deviations away from the mean and the TFIDF benchmark log loss of 0.6008 is greater than 9 standard deviations away from the mean. Thus we can conclude that our best fit model is significantly stronger than our benchmark result.

V. Conclusion

In this project, we tried to solve the problem - “How similar are two questions?”. This specific problem was addressed as a 2016 Kaggle competition organised by Quora. The dataset we are going to used for our analysis is the Quora Question Pairs dataset (https://www.kaggle.com/quora/question-pairs-dataset) which consisted of about 400K question pairs.

Our benchmark model was to convert the questions to word vectors with word weights given by TFIDF and then compute the cosine similarity between the two questions. A logistic regression model was used to determine the cosine similarity threshold above which we would predict the questions to be similar. This model gave us an accuracy of 0.6652 and log_loss of 0.6008 for 35000 training samples.

To improve on our benchmark model, a recurrent neural network architecture. We used GloVe.6B word vector embeddings to convert question word lists to vector representations. This was then passed on through a siamese LSTM layer which is then merged using L1 distance and then passed through a single unit sigmoid activated dense layer to predict outputs. Our best fit model has a LSTM layer of 70 units (100K parameters) with a dropout rate of 0.2. For 35000 training samples, we get a test accuracy of 0.7388 and log loss of 0.5310 which is better than our benchmark model.

Our LSTM model is a significant improvement on our benchmark model. But our accuracy to predict whether two questions are similar or not still has a log-loss of 0.53. The Kaggle on which this project is based on has ensemble deep neural network winning models with log-loss less than of 0.15 [https://www.kaggle.com/c/quora-question-pairs/leaderboard]. Thus, we know further improvements can be made to our models such as -

- Adding more features - e.g. different distance measures between the two word vectors such as cosine distance, Jaccard distance, length of word lists, etc.
- Training a bigger dataset - larger than our current training pairs of 35000. Increasing our training dataset from 7000 to 35000 already improved our log loss from 0.568 to 0.531. Running larger training sets should improve our models.
- Experimenting with other deep networks and ensemble models. Ensemble models improve the overall accuracy by fitting to different parts of the solution space. But training ensemble models require significant computational resources.
- Training embeddings for words which have embedding missing - About 29% of the words in our Embedding matrix have null embeddings. Training the parameters in the embedding layer (which needs significant computational resources) would improve our models.

References

1. Kaggle kernels by Lystdo (https://www.kaggle.com/lystdo/lstm-with-word2vec-embeddings).
2. Homma et al. 2017 (https://web.stanford.edu/class/cs224n/reports/2748045.pdf)
3. Bogdanova 2015 (https://aclweb.org/anthology/K15-1013)
4. Addair 2016 (https://web.stanford.edu/class/cs224n/reports/2759336.pdf)
5. http://ruder.io/optimizing-gradient-descent/index.html#nadam
6. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
7. http://cs229.stanford.edu/proj2015/054_report.pdf