Capstone Project

Ankur Saikia
05-NOV-2017

Quora Question Pairs

Definition

Project Overview

Quora is a place to gain and share knowledge. It is a platform to ask questions which are answered by the community itself. Being such a large community, related questions are often asked by members. Multiple questions with the same intent can cause seekers to spend more time finding the best answer to their question, and make writers feel they need to answer multiple versions of the same question. Quora values canonical questions because they provide a better experience to active seekers and writers, and offer more value to both groups in the long term. It is because of this reason Quora hosted the question pair challenge on Kaggle where they provided a train and test dataset consisting of similar and different question pairs. The expectation from the Kaggle community was to build state of the art models to predict whether the question pair is similar.

The dataset for the problem can be found in the competition page on kaggle but Quora has also released a updated dataset for the same problem (http://qim.ec.quoracdn.net/quora_duplicate_questions.tsv). This dataset is publicly available and we will use this to tackle the problem. The training dataset consists of question pairs and the target class for that pair (0 for non-duplicate pair and 1 for duplicate pair). It contains 404290 rows of data, of which 63% rows are non-duplicate pairs and 37% are duplicate pairs. Hence, we will divide the training set into two parts: train and test. We will use sklearn function train_test_split to do the splitting. We will use the train set to train and cross validate our model. We will measure the performance of our model using the test set.

Problem Statement

Predicting question pair similarity is a classification problem. It is a well-known problem in the field of Natural Language processing. We already have many state of the art models that attain high accuracy developed by the participants of the challenge on Kaggle. Before the challenge Quora was using Random Forest for classifying the question pairs. This problem can be approached in many ways and there are well known solutions based on boosting, others based on deep neural networks and many more.

We will use a deep neural network architecture for this problem. We will use keras which is a deep learning library in python. It provides us with many predefined functions to preprocess text e.g. Tokenizer() and text_to_sequences() etc. We start by removing the alphanumeric characters and converting short form of words to their full forms. We will use regular expressions for this purpose. Now we will convert our text to vectors using already available keras functions. Then we will feed the vectors to a neural network architecture for the classification task. We will use Embedding layers with pretrained weights, LSTM and Convolution 1D layers as the starting layers. The weights are calculated using embedding matrix created with the help tokenized words in the dataset and with the pretrained word embeddings model: stanfordNLP's GloVe: http://nlp.stanford.edu/data/glove.840B.300d.zip. These layers extract useful semantic information using the word vectors and their arrangement. On top of these layers we will use Dense Fully connected layers for our predictions. The final layer will give us the predicted probabilities which we will use to predict whether the question pairs are duplicate or not. We will then check the accuracy score to improve our model and parameters. The training, cross validation and testing will be done on different mutually exclusive subsets of the data and those will be prepared using python libraries such as numpy, pandas and scikit-learn. We will save our deep learning models using Keras Checkpoint functionality, that saves our model weights using h5py.

Metrics

We will use Accuracy as our evaluation metric for this problem. We already know the validation accuracy of our benchmark model which is 82.5%. Accuracy score would be most suitable for this problem because the dataset is not too skewed (even though it is not 50-50 split for positive and negative) and because precision and recall need equal weightage.

Analysis

Data Exploration

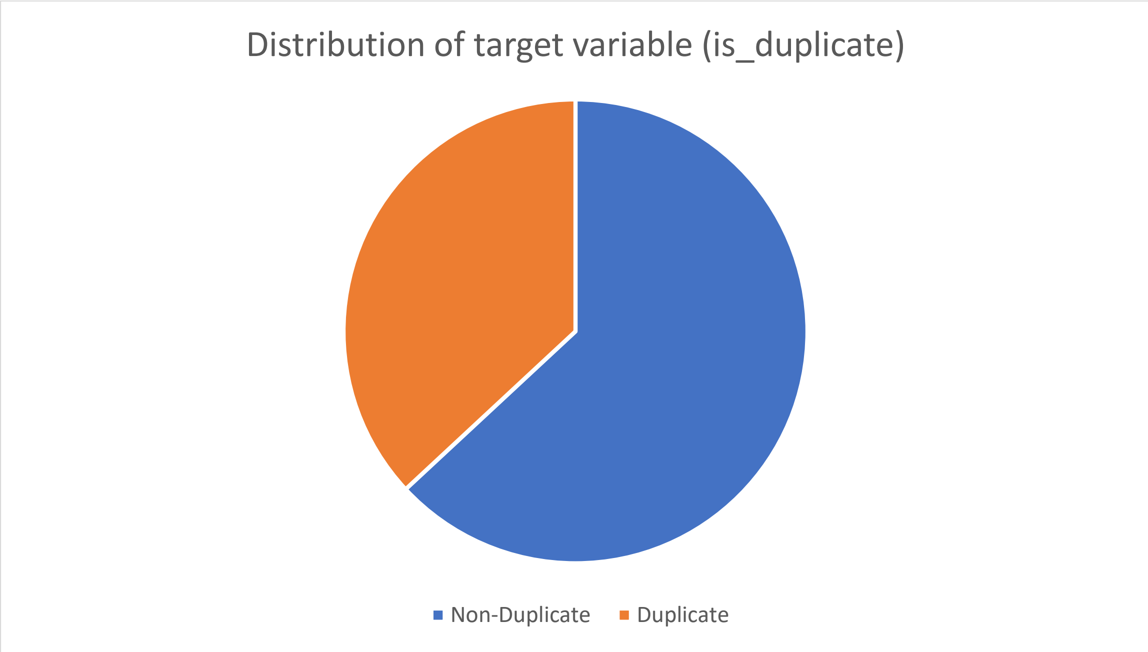
The dataset consists of 404290 rows and 6 columns. The column names are: 'id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate'. 'id' is the index column, 'qid1' and 'qid2' give the question id's and finally 'question1' and 'question2' are the questions itself. 'is_duplicate' is the target column that we need to predict. Of the 404290 rows present in the dataset, 255027 rows are for non-duplicate questions and 149263 rows are for duplicate questions.

Question 1 column has an average of 10.9 words per row. And Question 2 has an average of 11.2 words per row.

Examples:

Question1	Question2	Is_duplicate
What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	No
How can I be a good geologist?	What should I do to be a great geologist?	Yes

Exploratory Visualization



Algorithms and Techniques

We will use Keras and Tensorflow to train our neural network. Keras provides many high-level functions to pre-process our data and train our neural network with ease. For preprocessing we will use Regular Expressions. We will also use keras.Tokenizer() function to tokenize and convert out text data to vector sequences. We will also use sklearn to split our data to train and test sets. The important Keras layers and terms that we use in our models are:

- **Embedding layer:** This layer turns positive integers (indexes) into dense vectors of fixed size. This layer can only be used as the first layer of a model.
- **TimeDistributed Layer:** This wrapper applies a layer to every temporal slice of an input. In other words, it applies a layer to all the time steps present in the data. We can use any layer with the Time Distributed wrapper. For our models, we will use TimeDistributed with Dense layers
- **LSTM:** LSTM stands for Long Short Term memory and is a kind of Recurrent Neural Network. An LSTM is well-suited to classify, process and predict time series given time lags of unknown size and duration between important events.
- **Conv1D:** This is a Convolution 1D layer. This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs.
- **Dense:** This layer is the regular densely connected Neural Network layer.
- **Dropout:** This layer applies dropout to the input. Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.
- **BatchNormalization:** This layer normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

- **Binary-Crossentropy:** It is a type of loss function used in binary classification problems. A loss function (or objective function, or optimization score function) is one of the two parameters required to compile a model.
- **ReLU:** Relu stands for Rectified Linear Unit and is a type of activation function. The activation function of a node defines the output of a node given an input or set of inputs.
- **Sigmoid:** Sigmoid is also an activation function that has the characteristic 'S'-shaped curve.
- **Adam:** Adam is a kind of optimization algorithm that is used commonly in neural networks.

Benchmark

There are many state of the art models to tackle this problem. We would not be considering the models that were created using ensembles of many different models. Instead we will consider a single model as the benchmark that has been published on Medium: Implementing MaLSTM on Kaggle's Quora Question Pairs competition (https://medium.com/@eliorcohen/implementing-malstm-on-kaggles-quora-question-pairs-competition-8b31b0b16a07). This model scores around 82% accuracy on the validation set.

Methodology

Data Preprocessing

After importing the dataset as a data-frame, we remove symbols and punctuation marks from Question1 and Question2. For this we define a function clean_data() which takes some unprocessed input text and returns the processed text. The function removes punctuation marks such as: !,?,. etc. The function also fills up commonly used contractions such as: 'can't' gets replaced with 'can not', 'I'm' gets replaced with 'I am' etc. we take this preprocessing function from our Baseline model's implementation on github. (https://github.com/eliorc/Medium/blob/master/MaLSTM.ipynb)

After this step we call keras function Tokenizer() and fit it on our texts. This function lets us convert our text data to sequences using text_to_sequences(). We need to set the maximum length for our sequences. For this problem, we set the max length as 30. Also, there will be questions which do not have atleast 30 words, and hence they would have sequences of lesser length. For those questions we use the pad_sequences() function to fill the sequences with zeros and make their length 30.

Implementation

After pre-processing the data, we save the data-frame using pickle, so that we can directly use the pre-processed data.

We import the 'glove.840B.300d.txt' file downloaded from StanfordNLP website and save the data to a dictionary. The file consists of words represented by vectors of length 300.

We now take the complete set of words present in our dataset (collected using Tokenizer()) and assign glove vectors to each word thus creating a matrix (embedding_matrix). We will use this matrix to initialize our Embedding layer in Keras. We save this embedding_matrix using pickle so that we can directly reuse it later.

After this step, we start building our Keras model. At this step, I thought using Keras Sequential model would be a good option, but later realized that Keras 2.0 does not support the Merge of two Sequential models together. Hence, I used Keras Functional Api instead.

The first base model consists of two inputs: question1 sequence and question2 sequence and a single output that gives us the predicted class. The loss function used is 'binary-crossentropy' and optimizer is 'adam'. Two call-backs are also used in the model. The first call-back is 'Checkpoint' which will save the model weights with the best accuracy during training. The second call-back used is 'EarlyStopping' which will stop the training process when validation loss does not decrease by 0.0001 in 20 iterations.

For cross-validation, I thought of creating separate datasets, but later found out that Keras model.fit() method already has an option to use out of bag validation data. Hence, I went forward with that and used 10% of the training data as validation. Also, shuffle option was set as True to curb overfitting.

Refinement

For the base model, we start with a Keras Embedding layer and its weights were initialized by the embedding_matrix that we created before. The number of nodes for this layer was 300 and its weights were non-trainable. We then add a TimeDistributed Dense layer on top of it having 300 nodes. The third layer used is a Lambda layer, which lets us add our own functions. For this layer, we sum our tensors along the column axis.

The above setup is used to create two models 'q1' and 'q2', which are for question1 and question2. We now add the outputs of both the models to create a third model 'merged'. The output of 'merged' are then BatchNormalized and then fed to a Dense layer with 300 nodes. A Dropout of 0.2 is also added to layer. We then add another Dense layer with 300 nodes and a Dropout of 0.2.

All the Dense and TimeDistributed layers till this point were activated using 'relu'. We now add our final Dense layer with a single node and this layer is activated using 'sigmoid'.

We now compile our model using 'binary-crossentropy' as loss, 'adam' as optimizer, and 'accuracy' as the metric. We also define a Checkpoint call-back, which will save the model with the highest validation accuracy and a EarlyStopping call-back, which will monitor our validation loss and stop training if it does not improve by 0.0001 in 20 iterations.

We now start training our model using model.fit() on the training set and using a batch size of 500, we set the number of iterations as 100 and a validation split of 0.1 (10% of the testing data).

This model gets us upto 81.5% accuracy on the validation set and weights were saved as base 1.h5.

Now, for the second base model we increase our Dropouts to 0.3, and we use MAX instead of SUM in the Lambda layers. We increased the dropout because there were tell-tale signs of overfitting in the first base model (for the best iteration, train accuracy was 88.7 and validation accuracy was 81.5). The second model only worsened our predictions and the best iteration gave us an accuracy of 80.2%.

For the third model, we still use MAX in the Lambda layers but we decrease the dropout to 0.1. This time our predictions improved and the best iteration gave us 82.7% accuracy which is already higher than our benchmark.

Now, let us dive into CNN architectures. Our first CNN model consists of an Embedding layer initialized by the GloVe weights and this layer is not trainable. We then use a Convolutional1D layer with 64 filters, kernel_size of 2 and strides of 1. We use padding as ‘same’. We also add a dropout of 0.2 to the Convolution layer.The convolutional layer is followed by GlobalAveragePooling layer that will output the average of our filters as a 1D tensor. This architecture is used for both question1 and question2 vectors.

We now concatenate the outputs of the above architectures for question1 and question2 and add a fully connected layer to it. The fully connected layer consists of two Dense layers, first layer with 128 hidden units and second layer with 64 hidden units. A dropout of 0.1 is added to both the layers. These two layers are followed by a Dense layer with single node. The convolutional layers and the Dense layers excluding the final layer were activated using ‘relu’ and the final layer had a ‘sigmoid’ activation.

We used the same process for training and saved the best weight to ‘base 4.h5’. This architecture could only give us 81.5% accuracy but overfitting was a lot less compared to the base 3 model.

The final base model we will consider is with LSTM. Like before we start with an Embedding Layer initialized by GloVe weights. Now we add an LSTM layer with 30 nodes and set return_sequences to True so that the layer outputs all the sequences. This arrangement is done for both question1 and question2.

We now take the dot product of the outputs of the LSTM layers for question1 and question2, Flatten the resultant sequence and add two Dense layers on top of it. The Dense layers are accompanied by dropout of 0.1 and Batch Normalization. These dense layers are activated using ‘relu’. We now add the final Dense layer with 1 node and ‘sigmoid’ activation.

This model was also not sufficient to beat our benchmark.

So, we will use our third base model and refine it to get better results. This will be our final model. Our final solution is completely similar to Base 3 model only with minor changes that will be discussed in the Results section. This model gave us a final validation accuracy of 83.2% and an accuracy of 83.4% on the test set.

Results

Model Evaluation and Validation

The Final Model architecture is based upon Embedding layer initiated with GloVe Embedding weights, Time Distributed Dense layer, a lambda(max) layer and some fully connected dense layers. This model had 404,401 trainable parameters and the best accuracy was given in the 21’st iteration. The model weights are saved in ‘real_merge_2.h5’ file. Lets dive into the details and arrangement.

```
____________________________________________________________________________________
Layer (type)                     Output Shape        Param #      Connected to
====================================================================================
==================
input_1 (InputLayer)             (None, 30)          0
____________________________________________________________________________________
input_2 (InputLayer)             (None, 30)          0
____________________________________________________________________________________
embedding_1 (Embedding)          (None, 30, 300)     26510400     input_1[0][0]
```

embedding_2 (Embedding)	(None, 30, 300)	26510400	input_2[0][0]
time_distributed_1 (TimeDistributed)	(None, 30, 300)	90300	embedding_1[0][0]
time_distributed_2 (TimeDistributed)	(None, 30, 300)	90300	embedding_2[0][0]
lambda_1 (Lambda)	(None, 300)	0	time_distributed_1[0][0]
lambda_2 (Lambda)	(None, 300)	0	time_distributed_2[0][0]
subtract_1 (Subtract)	(None, 300)	0	lambda_1[0][0] lambda_2[0][0]
batch_normalization_1 (BatchNorm)	(None, 300)	1200	subtract_1[0][0]
dense_3 (Dense)	(None, 200)	60200	batch_normalization_1[0][0]
dropout_1 (Dropout)	(None, 200)	0	dense_3[0][0]
batch_normalization_2 (BatchNorm)	(None, 200)	800	dropout_1[0][0]
dense_4 (Dense)	(None, 200)	40200	batch_normalization_2[0][0]
dropout_2 (Dropout)	(None, 200)	0	dense_4[0][0]
batch_normalization_3 (BatchNorm)	(None, 200)	800	dropout_2[0][0]
dense_5 (Dense)	(None, 200)	40200	batch_normalization_3[0][0]
dropout_3 (Dropout)	(None, 200)	0	dense_5[0][0]
batch_normalization_4 (BatchNorm)	(None, 200)	800	dropout_3[0][0]
dense_6 (Dense)	(None, 200)	40200	batch_normalization_4[0][0]
dropout_4 (Dropout)	(None, 200)	0	dense_6[0][0]
batch_normalization_5 (BatchNorm)	(None, 200)	800	dropout_4[0][0]
dense_7 (Dense)	(None, 200)	40200	batch_normalization_5[0][0]
dropout_5 (Dropout)	(None, 200)	0	dense_7[0][0]

batch_normalization_6 (BatchNorm)	(None, 200)	800	dropout_5[0][0]
dense_8 (Dense)	(None, 1)	201	batch_normalization_6[0][0]

==

Total params: 53,427,801

Trainable params: 404,401

Non-trainable params: 53,023,400

The inputs to the model are word sequences of length 30 each. We have two inputs to the model, question1 sequence and question2 sequence respectively. These two sequences are fed into two Embedding layers which are non-trainable. The Embedding layers are initialized with GloVe embedding weights that we saved in the embedding_matrix. The outputs of the two embedding layers are then connected to two TimeDistributed Dense layers with 300 nodes each. We use 'relu' activation on the TimeDistributed Dense layers. The outputs are now connected to a Lambda layer which in return outputs the maximum along the column axis.

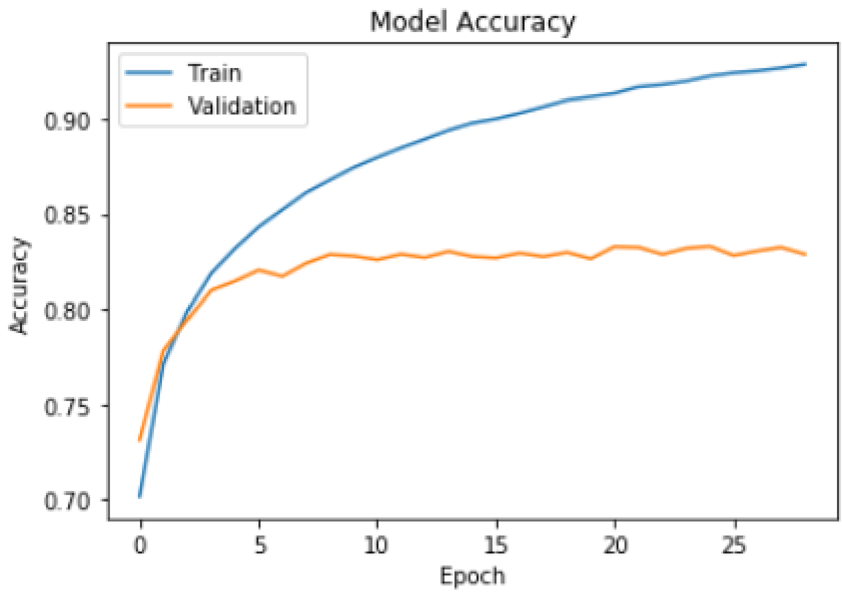
The outputs for question2 layers are now subtracted from the outputs from question1 layers, which helps us generate a merged model. We now add a BatchNormalize layer to the merged output. After this we add two Dense layers each with 'relu' activation, Dropout of 0.5 and outputs BatchNormalized. We then add three more Dense layers each with 'relu' activation, Dropout of 0.1 and outputs BatchNormalized. Finally, we add a Dense layer with single node and 'sigmoid' activation. We compile the model with 'binary-crossentropy' loss and 'adam' optimizer. After this we train the model on the training set for 50 iterations and save the model with the best accuracy using checkpoint. Our best iteration occurs at 21'st iteration and it gives a cross validation accuracy of 83.3%.

Justification

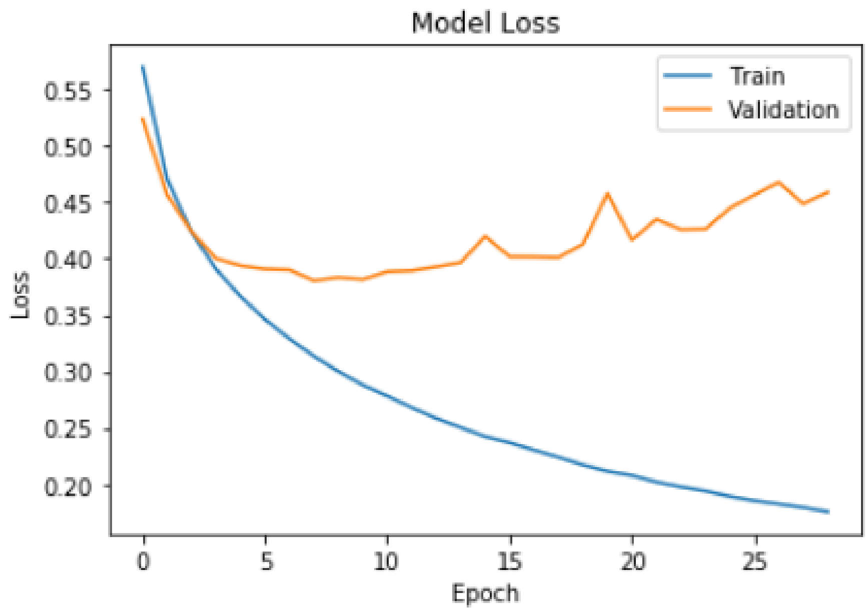
Since we only care about the Accuracy of the model, we see that our final model gives about 1% more accuracy than the baseline model. Considering our baseline model already was a state of the art model, an increase of 1% accuracy is a valid improvement. We also checked the predictions on the completely unseen test set and our model gave an accuracy of 83.4%.

Conclusion

Free Form Visualization



The above figure shows the increase in accuracy for training and validation set.



The above figure shows the variation in training and validation loss during the training.

Reflection

I started this project with a bit of research on the different architectures that are available for the question similarity problem. There are a vast number of well documented architectures available to tackle this kind of problem and that is where I learnt about TimeDistributed layers, Embedding layers, LSTM's and 1D Convolutions. This helped me create some base models and validate the performance.

Using the performance of the base models, I could select a architecture type and build on it to create the final model. I tried different arrangements for the final layer, adding more Dense layers, adding Dropouts etc. And finally reached at a model that gave us the best accuracy.

Improvement

There is a lot of room for improvement in this model. Some of the major areas where it can be improved are:

- Now, the model is overfitting and we can add regularization or increase Dropout to tackle it.
- We can train our own embeddings using only the words in our dataset.
- We can stack many different models to get even more accuracy and minimize the loss.