QUORA DUPLICATE QUESTION PAIR DETECTION

**Name of the College : SRM UNVERSITY**

**Student Name : T BHARAT**

**E-Mail ID : t.bharatt.bharat@gmail.com**

# Background

Quora is a question-and-answer site where questions are asked, answered, edited, and organized by its community of users

The heart of Quora is questions — questions that affect the world, questions that explain recent world events, questions that guide important life decisions, and questions that provide insights into why other people think differently. Quora is a place where you can ask questions you care about and get answers that are amazing

Quora provides users with platform to ask question that other user on site may answer.

However many questions that are asked at any given time have already been asked by others. Quora has only one version of each question. It doesn’t have a left wing version, a right wing version, a western version, and an eastern version.

Some of the best way to approach the solution set is by:

1. Deep learning models with Word Embeddings:

a. LSTM

b. 1D Convolutional Neural Nets

c. Keras with time distributed layers

2. XGBoost / Random Forest / Extra Trees with good feature engineering

# Our Understanding

Quora duplicate data set states that there will be no duplicate data in the data set as every question could not be answered to the user. So there are many duplicate question which are unnecessary and space consuming.

So here we are proposing a solution set to detect if the given two question are duplicate or not.

The data consisted of 404351 question pairs with 255045 negative samples (non-duplicates) and 149306 positive samples (duplicates).

Now in here we have to provide the solution. The model through which I am going to approach is by KERAS way.

For sentence similarity we have to find by Siamese Recurrent Architectures. It offers a pretty straightforward approach to the common problem of sentence similarity.

# Scope

**Elements being considered in the Solution**.

* 1. Pandas for preprocessing::

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

* 1. Keras for Networks::

As we are using Neural networks .Keras is optimum one and best to implement.

**3**. Siamese network for Sub network ::

Siamese networks are networks that have two or more identical sub-networks in

them .

Siamese networks perform well on similarity tasks and have been used

for tasks like sentence semantic similarity, recognizing forged signatures and many

more.

**4.** Adadelta optimizer as optimizer ::

The method dynamically adapts over time using only first order information and has

minimal computational overhead beyond vanilla stochastic gradient descent. The

method requires no manual tuning of a learning rate and appears robust to noisy

gradient information, different model architecture choices, various data modalities

and selection of hyperparameters.

**5.** Embedding matrix ::

In Embedding matrix we will assign each word its word2vec embedding and leave the unrecognized ones (less than 0.5%) random.

**6.** Data preparation with MaLSTM ::

In order to prepare our data for use in Keras we have to do two things:

* Split our data to ‘left’ and ‘right’ inputs (one for each side of the MaLSTM)
* Pad all of the word number sequences with zeros

**7.** Manhattan LSTM (MaLSTM) as Model ::

There are two networks LSTMa and LSTMb in MaLSTM . This model will help us as a predictor of semantic similarity.

**8.** Matplotlib as graph display ::

` The plot diplay we have displayed in the solution are done using matplotlib .

# Out of Scope

**Elements not being considered in the solution**

The detailed explaination of Siamese Recurrent Architectures have not been discussed in the given solution set.

There is only summery of the model been explained in the upcoming sessions

The optimizer is also implemented in the code .

There is also a summary of the LSTM and optimizer.

# Assumptions

**General Assumptions**

**Technical Assumptions**

# Solution Approach

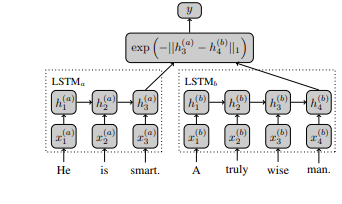
**High Level Solution Approach**

The solution to Quora solution is basically dependent on the Manhattan LSTM (MaLSTM)

a Siamese deep network and its appliance to Kaggle’s Quora Pairs competition.

Long Short-Term Memory (LSTM) network for labeled data comprised of pairs of variable-length sequences. Our model is applied to assess semantic similarity between sentences, where we exceed state of the art, outperforming carefully handcrafted features and recently proposed neural network systems of greater complexity. For these applications, we provide wordembedding vectors supplemented with synonymic information to the LSTMs, which use a fixed size vector to encode the underlying meaning expressed in a sentence (irrespective of the particular wording/syntax)The proposed Manhattan LSTM (MaLSTM) model is outlined in Figure .There are two networks LSTMa and

LSTMb which each process one of the sentences in a given pair, but we solely focus on siamese architectures with tied weights such that LSTMa = LSTMb in this work. Nevertheless, the general untied version of this model may be more useful for applications with asymmetric domains such as information retrieval (where search queries are stylistically distinct from stored documents).



We get the data as raw text, so our first mission is to take the text and convert it into lists of word indices. We first open the training data files in pandas The training data is stored in train\_df and the test data in test\_df and both are pandas DataFrames. These are the pre-processing steps with Pandas.

Now our aim is to have the ability to turn a word into its embedding given by word2vec, in order to do that we will need to build:

**1.** Vocabulary which is a dict where the keys are words (str) and values are the corresponding indices (a unique id as int).

**2.** Inverse\_vocabulary which is a list of words (str) where the index in the list is the matching id (fromvocabulary).

We also use gensim. models. KeyedVectors to load the word2vec embeddings.

Throughout the code only 2 functions of this class will be used, .vocab which will hold all of the word2vec words and .word\_vec(word) which takes a word and returns its embedding.

Finally we will use nltk's English stopwords and store them in stops.

Our next goal is to create the embedding matrix.

We will assign each word its word2vec embedding and leave the unrecognized ones (less than 0.5%) random.

# Technical Architecture

**Proposed Architecture**:

Manhattan LSTM (MaLSTM) Siamese deep network

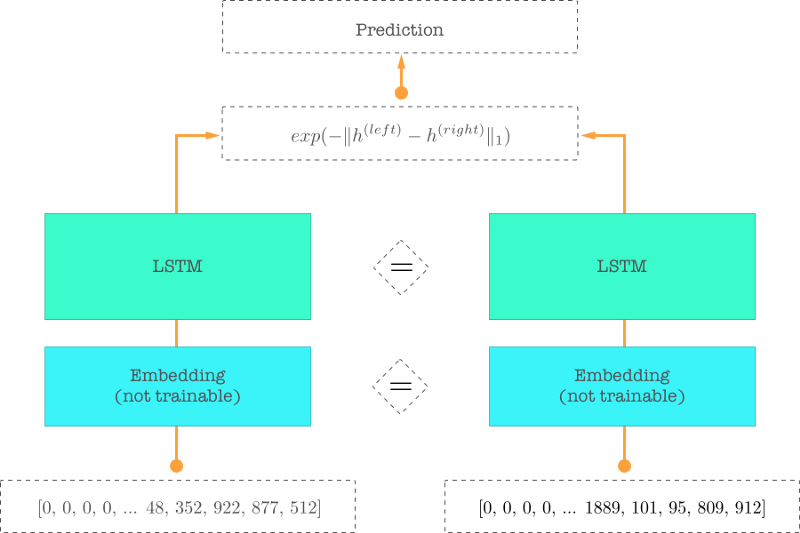
It offers a pretty straightforward approach to the common problem of sentence similarity. Named MaLSTM (“Ma” for Manhattan distance), its architecture is depicted in figure 1 (diagram excludes the sentence preprocessing part). Notice that since this is a Siamese network, it is easier to train because it shares weights on both sides.

In the following figure MaLSTM’s architecture — Similar color means the weights

are shared between the same-colored elements.

Siamese networks are networks that have two or more identical sub-networks in it.

Siamese networks seem to perform well on similarity tasks and have been used for tasks like sentence semantic similarity, recognizing forged signatures and many more.



In MaLSTM the identical sub-network is all the way from the embedding up to the last LSTM hidden state.

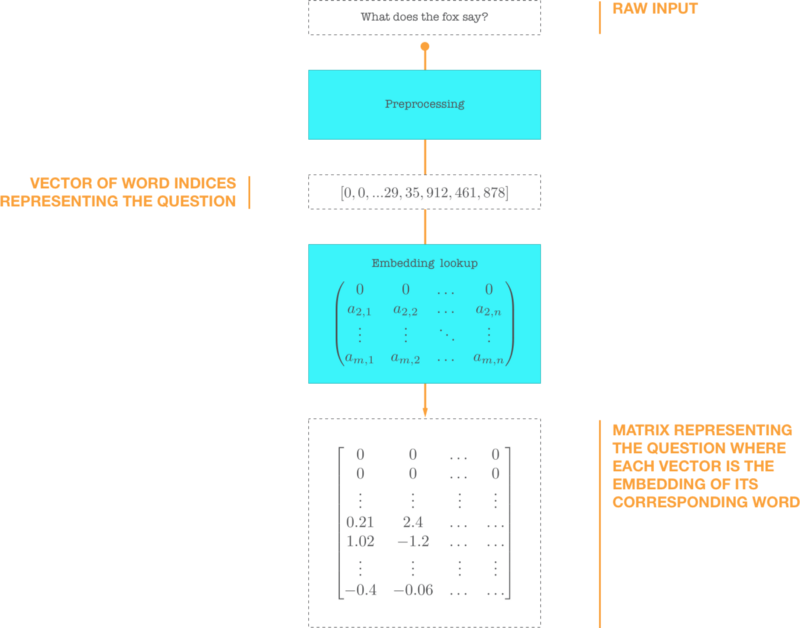
Word embedding is a modern way to represent words in deep learning models

Essentially it’s a method to give words semantic meaning in a vector representation.

Inputs to the network are zero-padded sequences of word indices. These inputs are vectors of fixed length, where the first zeros are being ignored and the nonzeros are indices that uniquely identify words.

Those vectors are then fed into the embedding layer. This layer looks up the corresponding embedding for each word and encapsulates all them into a matrix. This matrix represents the given text as a series of embeddings.

The process is depicted below.



We have two embedded matrices that represent a candidate of two similar questions. Then we feed them into the LSTM (practically, there is only one) and the final state of the LSTM for each question is a 50-dimensional vector. It is trained to capture semantic meaning of the question.  
In previous figure this vector is denoted by the letter h.

By now we have the two vectors that hold the semantic meaning of each question. We put them through the defined similarity function and since we have an exponent of a negative the output (the prediction in our case) will be between 0 and 1.

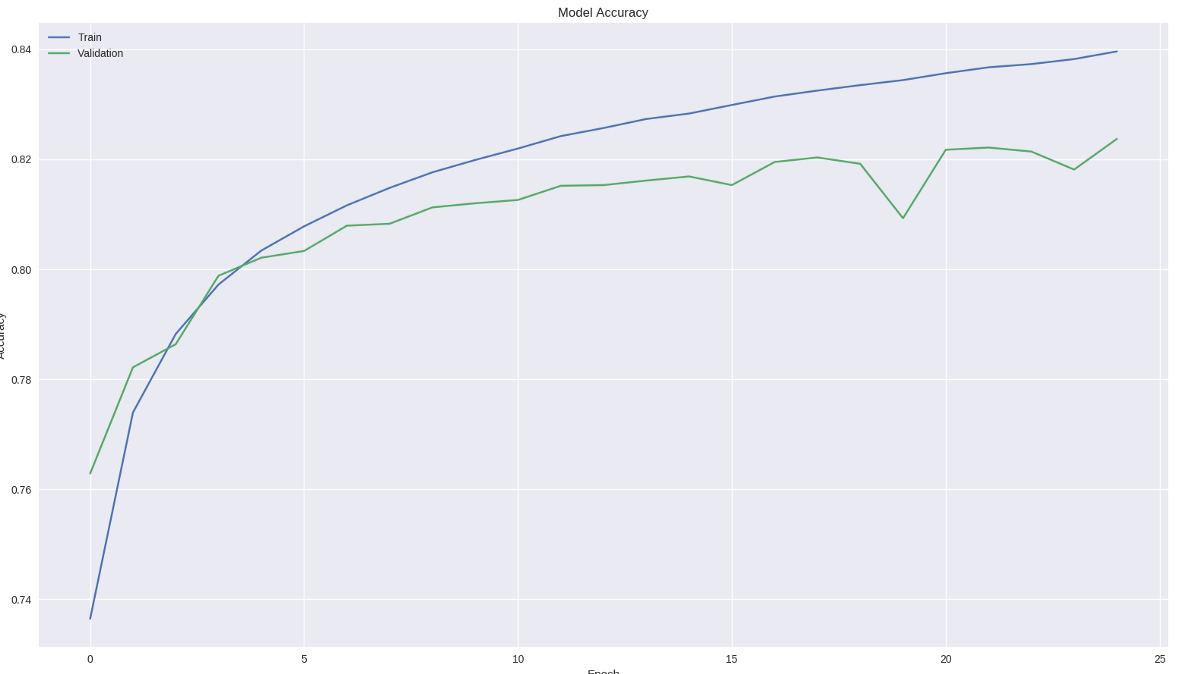
https://cdn-images-1.medium.com/max/800/1*W6AQLzU6yZqLc5pgcbMtMg.png

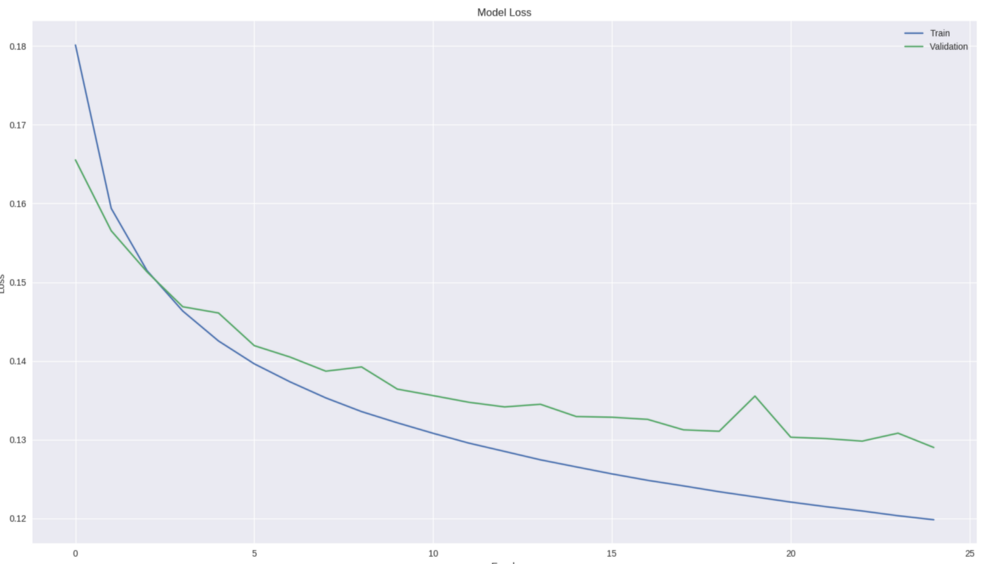
The explained model of MaLSTM in Keras has the

The whole script and including preparation and training took about 30 hours depending on the GPU configuration as of mine is NVIDIA 820.

Training data vs validation data accuracy and loss

**Accuracy:**

****

**Loss:** ****

There is about 80% accuracy on validation data.

# Implementation Framework

**General Implementation Approach**

* Kaggle's test.csv is too big, so I had extracted only the top 20 questions and created a file called test-20.csv and It is used in the predict.py
* Put all data files to ./data directory.
* **How to Run**

TRAINING:

$ python3 train.py

PREDICTION :

$ python3 predict.py

The accuracy is about 80%

**Hardware and Software Details**

The code runs properly in my NVIDIA 820 2 GB graphics

You can directly run on python with above commands of training and prediction.

Before that you have to put all data files to ./data directory.

OR

As I have done the installation and worked in Anaconda SPYDER IDE

conda install pandas

conda install numpy

conda install scikit-learn

conda install nltk

conda install tqdm

conda install keras

conda install tensorflow

conda install pyemd

conda install fuzzywuzzy

conda install python-levenshtein

conda install --upgrade genism

These are the required libraries in the conda environment

Same libraries you can install with pip too.

After the instllation of the libraries to the system then you can download the project from the github and directly run the python code.

1. Solution Submission

The solution is available in the given github link::

https://github.com/bharatsunny/TCS-QUORA-DUPLICATE-QUESTION-PAIR-DETECTION-

# References

* http://www.mit.edu/~jonasm/info/MuellerThyagarajan\_AAAI16.pdf
* https://keras.io/layers/wrappers/
* https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning
* https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
* https://stackoverflow.com/questions/47262955/how-to-import-keras-from-tf-keras-in-tensorflow