Quora Duplicate Question Pairs

Shraddha, Sunil (.8)
Jayavardhan Reddy, Peddamail (.1)

Abstract

Finding an answer to any question is becoming easier by the day but finding the "best" answer is getting increasingly difficult. Our project focuses on identifying questions with similar intent in question answering sites, specifically, the site Quora. Given a dataset consisting of question pairs, we try to classify the given pairs as duplicate or not duplicate. Identifying duplicate questions is not an easy task since questions that have no words in common could still have the same intent and questions with almost the same words might have different intent. We tried to tackle this problem through elegant feature engineering using one-hot features, questions length information, Word Embedding both Glove as well as Word2Vec embeddings. We implemented both simpler Machine Learning models like Random Forest, SVM, Logistic regression and also complex Neural Network models like CNN, LSTM, Bidirectional LSTM Networks using the Siamese architecture. The performance of models has been compared and a detailed analysis was performed about the different features and parameter tuning for the different models implemented.

Introduction

Problem Statement:

In this age of information, the answer to any question can be found in just a few clicks. One of the key parts enabling this instant knowledge access is the plethora of knowledge transfer platforms, also known as question-answering sites. One of the rising issues faced by these sites is the problem of data duplicity, where many variations of the same questions make it hard to find the best answer.

For the project, the focus is on Quora, a question-and-answer site where questions are asked, answered, edited and organized by its community of users. Quora has millions of users, which leads to the presence of multiple questions which address the same issue. As an answer seeker, it is an onerous task to browse through all similarly worded questions to find the best answer. Also, answer writers often have to write the same answer for different versions of the same question. Identifying question duplicates will let users find the best answers in an easier way and have a better overall experience.

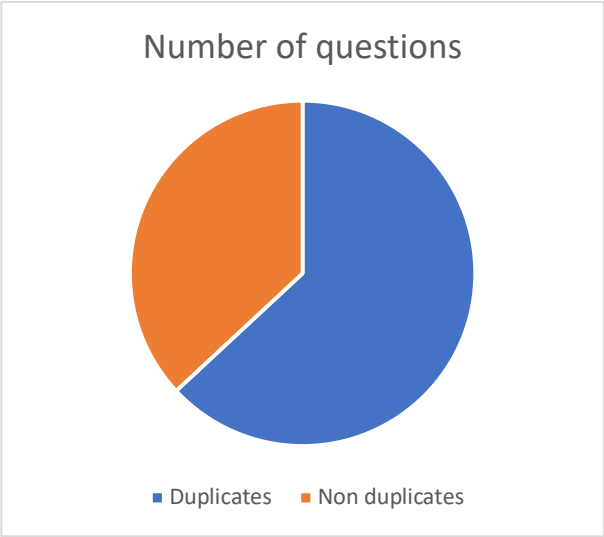
Identifying duplicate questions is same as determining whether two sentences have the same meaning. This is a difficult problem to solve. "What is the best way to lose weight?" and "How to become thin?", "What are you going to eat in the morning?" and "What will be your breakfast?" have the same meaning but do not have any words in common. Marking these questions as duplicate would require capturing the semantic relationship between the words and the context in which these words are used.

DataSet:

The datasets needed for this project have been obtained from the Kaggle site. The files are:

1. Train Set:
    a. Size - 21.17 MB
    b. Number of Question Pairs: 404290
2. Test Set:
    a. Size -112.47 MB
    b. Number of Question Pairs: 2345796

The training dataset contains a total of 404290 question pairs with a split up of 255026 duplicate question pairs and 149263 non-duplicate question pairs. It can therefore be seen that the training set is a bit skewed with 63% duplicate pairs and 37% unique question pairs.



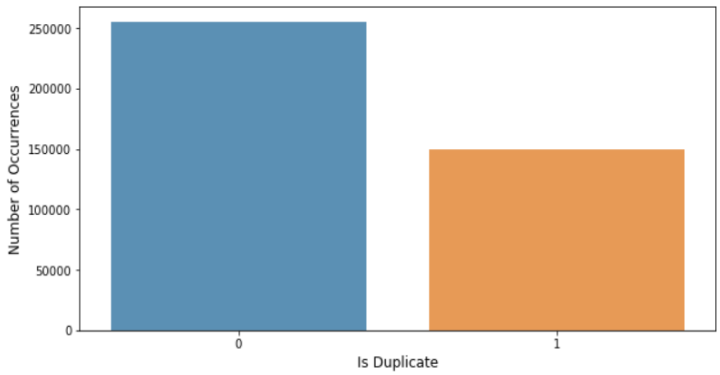
API's Used:

1. **Keras**: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.
2. **Scikit-Learn:** Its one of the first and most famous library for Machine Learning in Python. It is built on NumPy, SciPy, and matplotlib. Very easy to use and well documented.
3. **Gensim:** Gensim is a Machine Learning Library which has very good functionalities for LDA and Word2vec. Developed by Radim Rehurek, it is highly scalable and support parallel implementation on multiple cores.
4. **Plotly and Seaborn:** Plotly and Seaborn create leading open source tools for composing, editing, and sharing interactive data visualizations. They are better than the inbuilt plotting API of python(matplotlib), in terms of visual appeal as well as the variety of plots available.
5. **NLTK:** NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification,

tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.

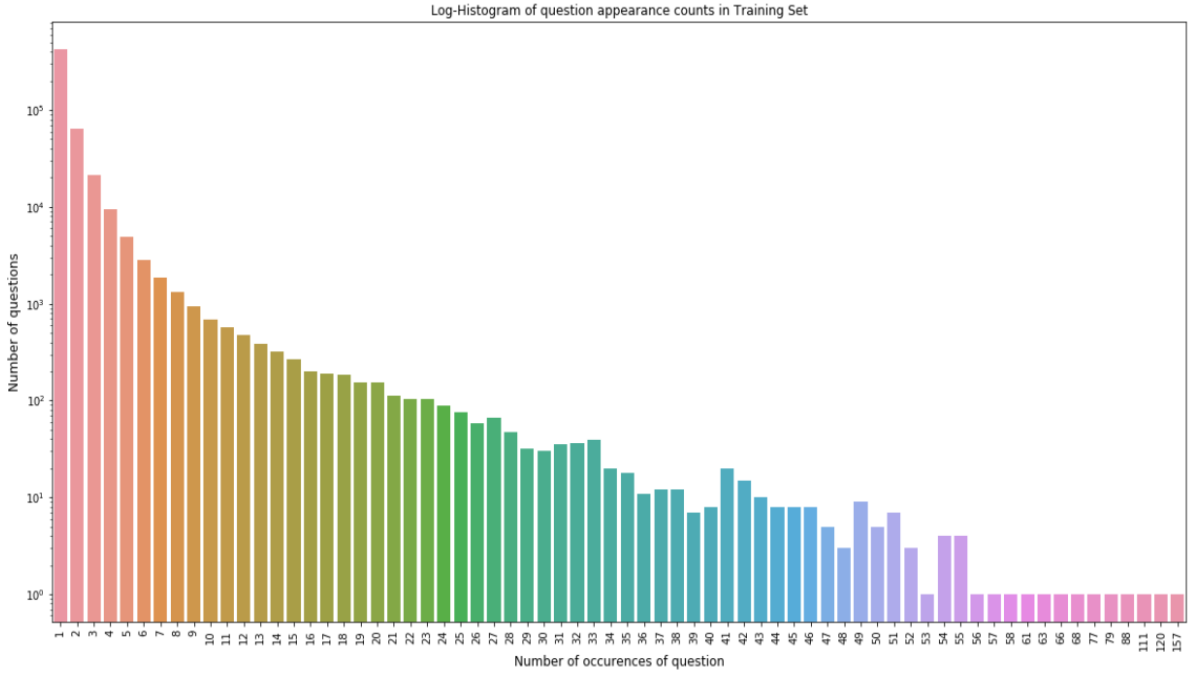
Exploratory Data Analysis

The training dataset contains a total of 404290 question pairs with a split up of 255026 duplicate question pairs and 149263 non-duplicate question pairs. It can therefore be seen that the training set is a bit skewed with 63% duplicate pairs and 37% unique question pairs.

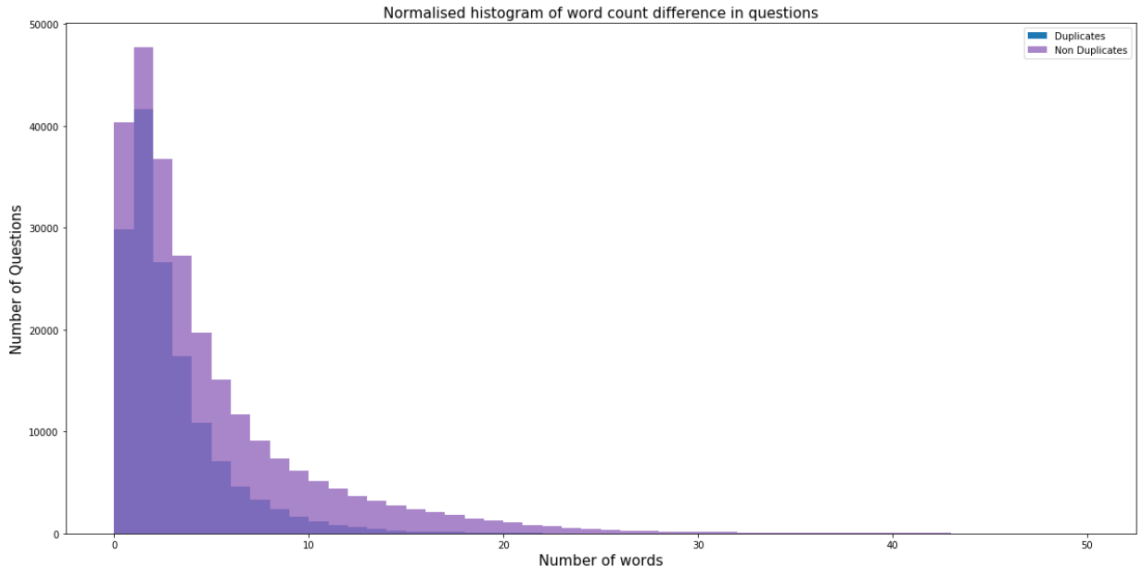


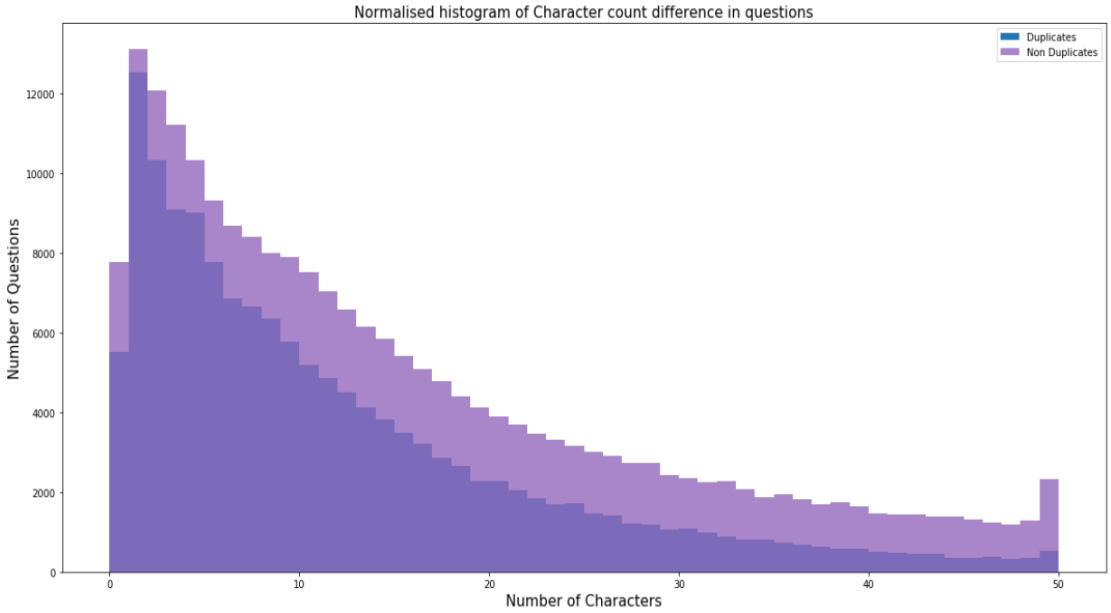
The test set consist of 2345796 question pairs but according to the information provided in the Kaggle site, most of the pairs in the dataset were autogenerated to keep a check on cheating. The total number of unique questions in the dataset is 537933 with most of the questions appearing just once in the dataset.

The graph below represents the number of unique questions and the number of questions that there are multiple occurences of, in the training dataset. It is found that there is a good distribute of number of questions vs number of occurences of every question in the dataset.

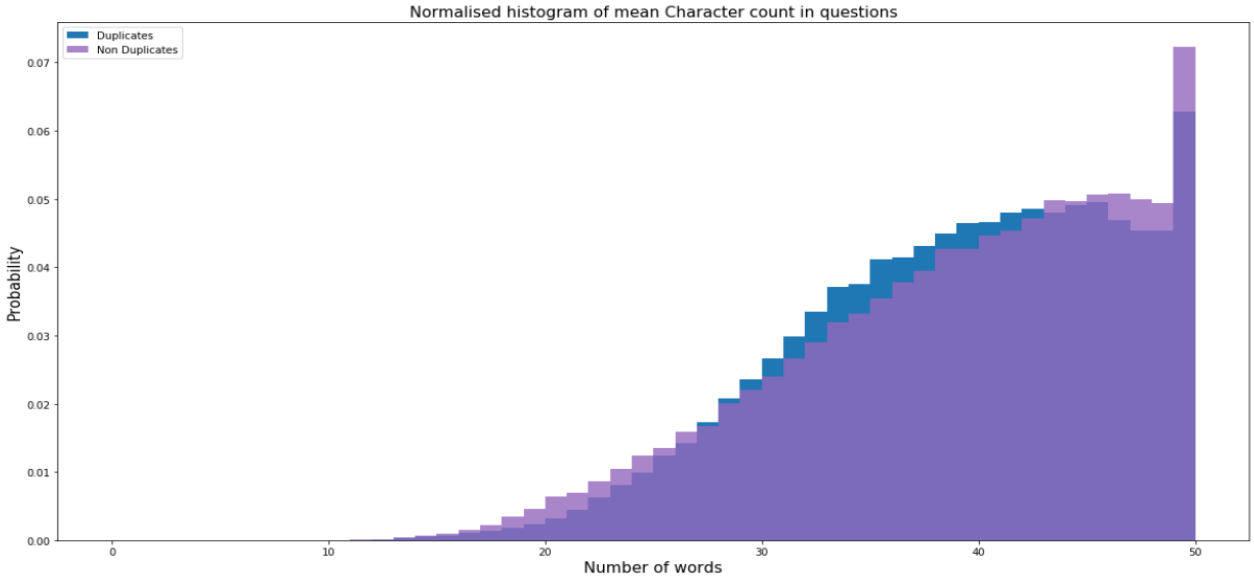


The graph below is a histogram representation of the number of questions vs the number of words in each question.
The average character count and word count of the questions were estimated. This could be a useful metric while deciding the pad size of question before feeding them as an input to the neural networks. It can be seen that the distribution is almost the same for both duplicates and non-duplicates.





The probability distribution was plotted for both the datasets instead of the histogram count to account for the difference in the size of the datasets. It can be seen that the distribution is almost the same for both training and test datasets



The Pearson correlation coefficient, also referred to as the Pearson's *r*, Pearson product-moment correlation coefficient (PPMCC) or bivariate correlation, is a measure of the linear correlation between two variables *X* and *Y*. It has a value between +1 and −1, where 1 is total positive linear correlation, 0 is no linear correlation, and −1 is total negative linear correlation.

The usefulness of features like Cosine similarity, Euclidean similarity and Manhattan distance is calculated to see if it can be used as an important feature for further analysis.

Word Embeddings

Word embeddings are used in most state of the art techniques for determining text similarity. A word embedding is a learned representation for text where words that have the same meaning have a similar representation. It is this approach to representing words and documents that may be considered one of the key breakthroughs of deep learning on challenging natural language processing problems. Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network. Each word is represented by a real-valued vector, often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as a one-hot encoding.

The distributed representation is learned based on the usage of words. This allows words that are used in similar ways to result in having similar representations, naturally capturing their meaning. This can be contrasted with the crisp but fragile representation in a bag of words model where, unless explicitly managed, different words have different representations, regardless of how they are used.

Advantages:

1. One of the benefits of using dense and low-dimensional vectors is computational: the majority of neural network toolkits do not play well with very high-dimensional, sparse vectors.
2. The main benefit of the dense representations is generalization power. Captures contextual similarity between words.

Types of Word Embeddings:

Word2Vec: Word2Vec is a statistical method for efficiently learning a standalone word embedding from a text corpus. It was developed by Tomas Mikolov, et al. at Google in 2013 as a response to make the neural-network-based training of the embedding more efficient and since then has become the de facto standard for developing pre-trained word embedding. Additionally, the work involved analysis of the learned vectors and the exploration of vector math on the representations of words. Word2Vec representations are surprisingly good at capturing syntactic and semantic regularities in language, and that each relationship is characterized by a relation-specific vector offset. This allows vector-oriented reasoning based on the offsets between words. For example, the male/female relationship is automatically learned, and with the induced vector representations, “King – Man + Woman” results in a vector very close to “Queen.”

Two different learning models were introduced that can be used as part of the word2vec approach to learn the word embedding; they are:

- Continuous Bag-of-Words Model: Predicting the current word based on its context.
- Continuous Skip-Gram Model: Predicting the surrounding words given a current word.

GloVe Vectors: The Global Vectors for Word Representation, or GloVe, algorithm is an extension to the word2vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford. Classical vector space model representations of words were developed using matrix factorization techniques such as Latent Semantic Analysis (LSA) that do a good job of using global text statistics but are not as good as the learned methods like

word2vec at capturing meaning and demonstrating it on tasks like calculating analogies (e.g. the King and Queen example above). Rather than using a window to define local context, GloVe constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus. The result is a learning model that may result in generally better word embeddings.

Pre-trained Word Embedding:

Training Word Embeddings is a computationally expensive task. We downloaded and used pre-trained word embedding for our project.

- Pre-Trained Word2Vec Embeddings:
    - Dimension: 300d
    - Number of Words: 3 million words
    - Training Set: 100 billion words from Google News dataset
- Pre-Trained GloVe Embeddings:
    - Dimension: 50d,100d, 200d, 300d
    - Number of words: 400k words
    - Training Set: 6 billion tokens from Wikipedia 2014 + Gigaword 5

We used 100d Glove Embedding for our analysis

Implementation of different models

Tf-idf is used as a feature for pre-processing. Tf-idf stands for *term frequency-inverse document frequency*, and the Tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

The following models were implemented using the sklearn library.

1. Random forest-Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.
2. Logistic regression- is a regression model in which the outcome is binary and the output predicts the probability from 0 to 1 of the outcome.
3. Decision trees-The decision tree algorithm tries to solve the problem, by using tree representation. Each internal node of the tree corresponds to an attribute, and each leaf node corresponds to a class label. The best attribute of the dataset at the root of the tree is chosen. The training set is divided into subsets in such a way that each subset contains data with the same value for an attribute. This process is repeated until all the leaf nodes are found

4. Naïve Bayes- Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.
5. SVM- An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible

Results:

Model	Training accuracy	Validation accuracy	Training loss	Validation loss
Random forest	0.775	0.721	0.685315	0.697242
Logistic regression	0.685	0.669	0.72543	0.741672
Decision tree	0.672	0.695	0.75232	0.74327
Naïve bayes	0.574	0.633	0.8472	0.8253
SVM	0.429	0.553	0.8723	0.8244

Neural Network Based Approaches

Siamese Neural Network Architecture:
Siamese neural network is a class of neural network architectures that contain two or more identical subnetworks. identical here means they have the same configuration with the same parameters and weights. Parameter updating is mirrored across both subnetworks.

Siamese NNs are popular among tasks that involve finding similarity or a relationship between two comparable things. Some examples are paraphrase scoring, where the inputs are two sentences and the output is a score of how similar they are; or signature verification, where figure out whether two signatures are from the same person. Generally, in such tasks, two identical subnetworks are used to process the two inputs, and another module will take their outputs and produce the final output.

Advantages of Similar Neural Networks:
1. Sharing weights across subnetworks means fewer parameters to train for, which in turn means less data required and less tendency to over fit.
2. Each subnetwork essentially produces a representation of its input. If your inputs are of the same kind, like matching two sentences or matching two sentences, it makes sense to use similar model to process similar inputs. This way you have representation vectors with the same semantics, making them easier to compare.

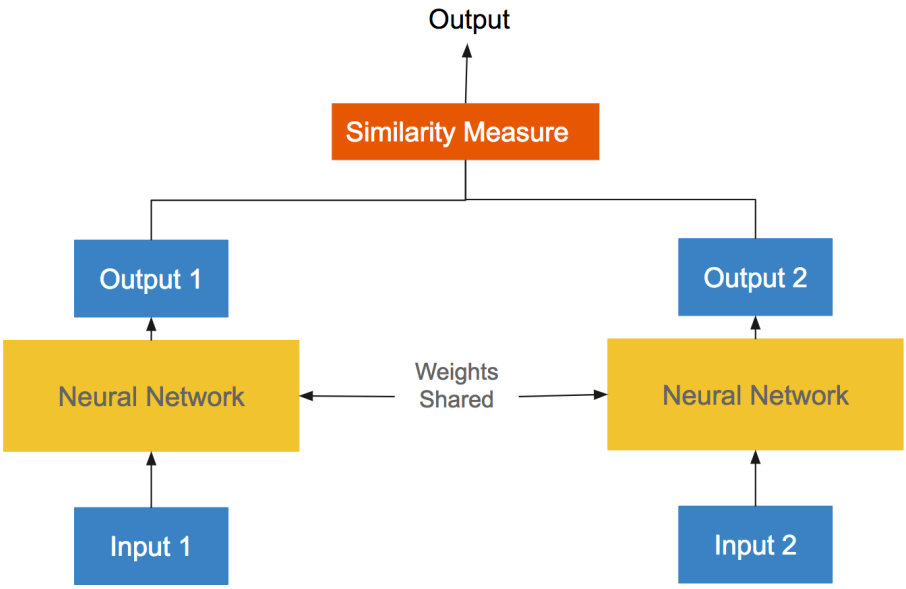


Fig: Siamese Neural Network Architecture

Siamese Architecture for Question Similarity:
For the task of Question Similarity, the input1 and input2 would be question1 and question2 respectively. The Neural Networks do not understand text as human do, so we need to convert questions (each word in the question) to a representation which a Neural Network can understand. Different types of encodings techniques are used for depending on task in NLP. We employed the Word Embedding representation to convert words to a representation understandable by the Neural Network.

Convolutional Neural Nets:
A convolutional neural network (CNN) is a feed forward artificial neural network in which the connectivity patterns between its neurons are inspired by the organization of the visual cortex. They are bio inspired processes that consist of variations of multilayer perceptrons designed to process data using minimal amount of preprocessing.

Convolutional neural networks have been used in a wide variety of applications such as image and video recognition, recommender systems and natural language processing.

In traditional neural networks, a single vector is received as an input and is transformed through a series of hidden layers. Each hidden layer is made up of a set of neurons where each neuron is fully connected to all neurons in previous layer and where neurons in a single layer function independently without sharing any connections. The last fully connected layer is called the 'output layer' and in classification settings it represents the class scores.

In convolutional neural networks, the input vectors go through a sequence of layers and every layer transforms one volume of activations to another through some differentiable function. The three main layers used in CNNs are convolutional layers, pooling layers and fully-connected layers.
In natural language processing (NLP), the input vector is first converted into numeric form using word2vec and gloVe. Just as the input in images is represented as pixels, the input in natural language processing is represented as a matrix where each row represents one-word token. For

example, for a 15-word sentence, we would have a 15 x 100 matrix if we have a 100-dimensional embedding.

The convolutional layer is the core building block of a CNN. This layer consists of a set of filters that extend through the entire depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product of the entries of the filter and the input and producing a 2-dimensional activation map. As a result of activations, some specific type of feature gets detected.

The output of the convolutional layer involves the stacking of the activation maps for all filters along the depth dimensions.

The hyperparameters that control the size of the output volume include: the depth, stride and zero-padding.

1. The depth of the output volume controls the number of neurons that connect to the same region in the input volume. Different neurons activate for different features in the input.
2. The stride controls how depth columns are allocated. When stride is 1, a new depth column of neurons is allocated to positions only 1 spatial position apart. This leads to heavily overlapping yields between the columns and larger volumes. When larger stride lengths are used, overlapping is much less and the output will be of a smaller volume spatially
3. The size of zero padding is the third hyperparameter that controls the output of volume spatial size.

Another important layer in a convolutional neural network is the pooling layer. There are several different pooling functions that can be implemented but maxpooling has been found to be the most common. It partitions the input matrix into a set of non-overlapping rectangles and for each sub-region, it outputs the maximum. The main purpose of the pooling layer is to reduce the spatial size of the representation which in turn reduces the amount of computation required and the number of parameters. Another purpose of the pooling layer is to control over fitting. It is common to periodically include pooling layers in between successive convolutional layers in a CNN architecture.

The pooling layer operates on every slice of the input and reduces it spatially. When a pooling layer of size 2x2 is used with a stride of 2, 75% activations get discarded.

Another type of pooling is average pooling or L2-norm pooling where instead of outputting the maximum within that slice, the average is outputted. Average pooling has proven to be not as useful as maxpooling.

Choosing the correct hyperparameters is crucial in optimizing convolutional neural networks. Since every feature map size decreases with depth, layers near the input will tend to have fewer filters while later layers tend to have more.

CNNs use more hyperparameters than a standard MLP. While the usual rules for learning rates and regularization constants still apply, the following should be kept in mind when optimizing convolutional networks.

Another important hyperparameter that has to be taken into consideration is the filter shape. Different filter shapes are can be used depending on the dataset in order to get the greatest possible set of abstractions.

Regularization in machine learning is a process of introducing additional information in order to prevent overfitting. One of the most widely used techniques of getting rid of over fitting is through dropout. At each training stage, individual nodes are dropped out of the network with probability (1-p) or kept with probability p.

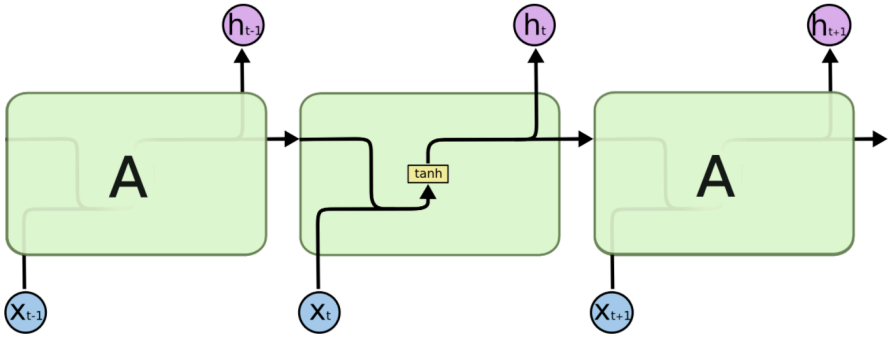
The probability that a hidden node will be retained is lower than the probability that an input node will be retained. The reason for this is obvious in that if an input node is dropped, information is directly lost.

One of the simplest methods to prevent overfitting of a network is to simply stop the training before overfitting has had a chance to occur. This technique comes with the disadvantage that the learning process is halted.

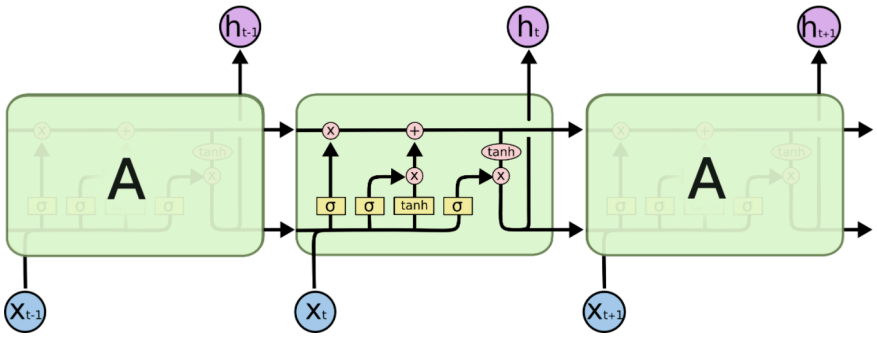
Another regularizer is weight decay which simply adds an additional error proportional to the sum of weights (L1) or squared magnitude (L2) of the weight vector, to the error at every node.

When using convolutional neural networks to solve problems in computer vision, there are two aspects that must be taken into account: Location invariance and compositionality. It can be assumed that there is an object in the image that must be identified. In this case, it does not matter where in the image that object occurs in order to be able to classify it accurately as the filters that are being used are going to be sliding over the entire image. The second aspect-compositionality implies that at every layer of the CNN, the filter converts lower-level features to higher-level representation. This means that shapes are detected from edges and objects are detected from shapes and so on. This works well in images, however in natural language processing, most of the learning happens in the first layer itself hence fewer layers can be used for applications in this area. However, the downside of using CNNs in NLP is that the order of the words matters in a sentence, unlike how it is in images. Taking these aspects into account, it would make intuitive sense to use recurrent neural networks to solve the same task.

Long Short-Term Memory Networks:
Long Short-Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They work tremendously well on a large variety of problems involving sequential data, and are now widely used. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure.



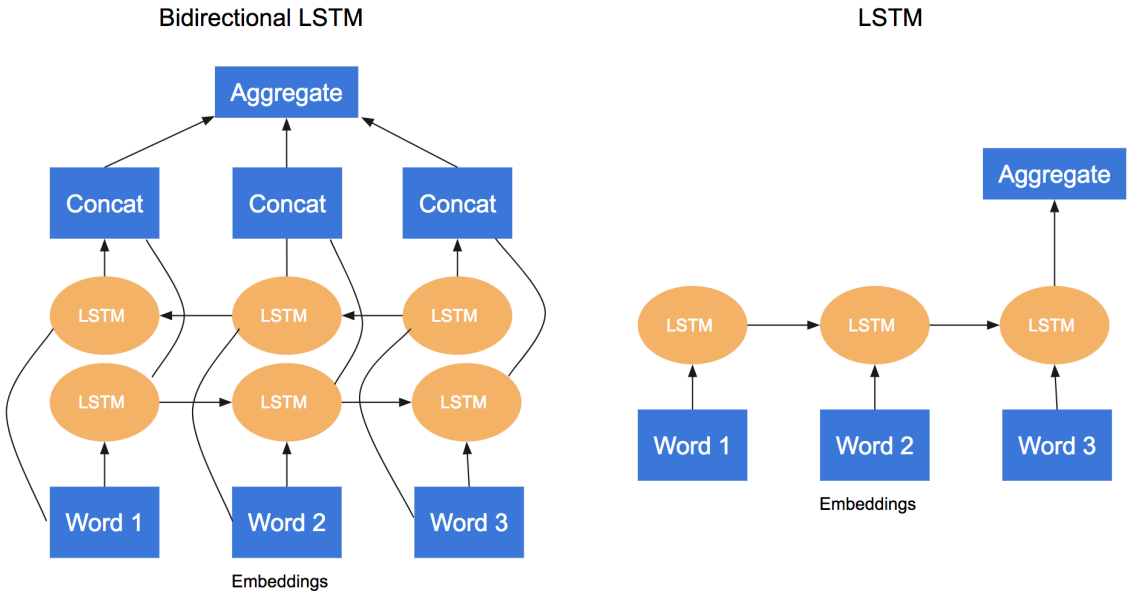
Simple RNN



LSTM

Bidirectional LSTM Network:

Bidirectional LSTM is a special kind of LSTM, where we have two LSTM layers, one LSTM layer trained in forward to backward sequence and other LSTM trained in backward to forward direction. In this way, one LSTM captures the forward dependencies and the other LSTM captures the backward dependencies. The outputs of both the layers are aggregated to produce the final representation.



Bidirectional LSTM Network

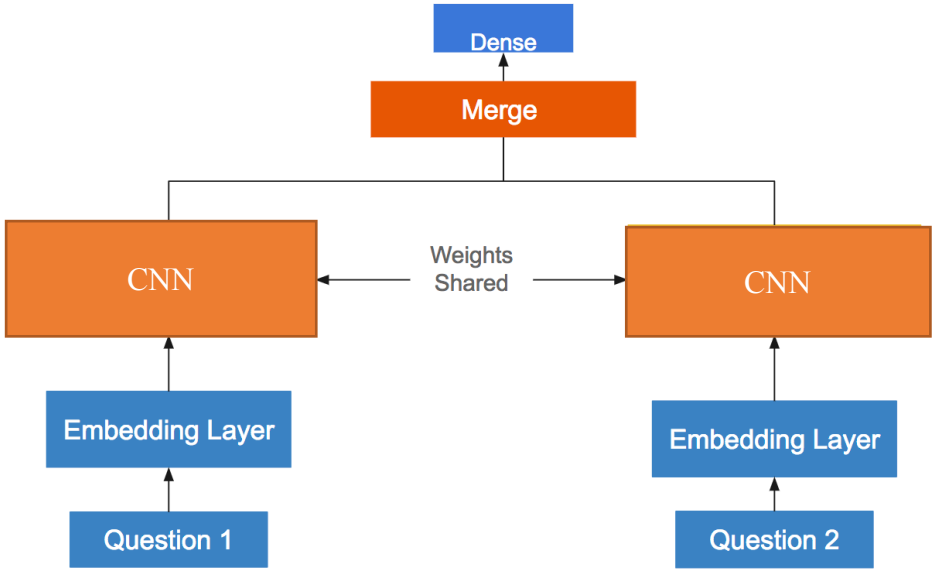
Model Design and Development

Siamese architecture was used has the template and we tried Convolutional Neural Networks, Long Short-Term Memory Networks and Bidirectional LSTM Networks.

Siamese Architecture with Convolution Networks:

The Model is designed using Keras Deep Learning library that has the. The layers of the model are stacked one after another Sequentially. Question1 and Question2 are given has input to the first layer separately and similarity measure is produced as output from the last year. The two questions act as separate inputs to the embedding layer.

Embedding Layer (First Layer): This layer is the first layer in Deep NLP Architectures. This layer maps a given input and converts it into its corresponding Word Embedding. The embedding layer is then fed into the convolution layer.
Conv1D layer: Convolution1d performs 1d convolution on the word embeddings that are obtained as the output of the embedding layer. When the convolutional layer is implemented, the parameters that need to be included are the number of filters, the kernel size, the stride length and the shape of the input.
Max-pooling layer: Max pooling is a sample-based discretization process. The objective is to down-sample an input representation, reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions. This is done to help prevent over-fitting as well as to reduce the computational cost by reducing the number of parameters to learn. The parameters that need to be set in the pooling layer are as follows: pool size- the size of max pooling windows, strides-the factor by which to downscale, padding-specify whether or not padding is required.
Dropout: Dropout is carried out on the output of the maxpooling layer**.** Dropout consists of randomly setting a fraction rate of input units to 0 at each update during training time which prevents overfitting. For example, if dropout is set at 0.2 then 2 out of units are dropped.
Flatten: The flatten large is used to flatten the output of the CNN layer.
Merge Layer: Merge Large is used to combine different vector outputs from the LSTM. Both LSTM's will produce a sentence vector, merge layer combined them to produce a single vector output.
Dense Layer: It is a regularly connected dense layer. It converts the vector output from the Merge Layer into a number between 0 and 1, which shows the measure of the similarity predicted by the network.



Siamese architecture using Convolutional Neural Networks

```
model - Siamese Networks
____________________________________________________________________
Layer (type)                 Output Shape          Param #
====================================================================
input_1 (InputLayer)         (None, 250)           0
____________________________________________________________________
input_2 (InputLayer)         (None, 250)           0
____________________________________________________________________
embedding_1 (Embedding)      (None, 250, 300)      116400
____________________________________________________________________
conv1d_1 (Conv1D)            (None, 246, 128)      192128
____________________________________________________________________
conv1d_2 (Conv1D)            (None, 246, 128)      192128
____________________________________________________________________
max_pooling1d_1 (MaxPooling1D)   (None, 49, 128)   0
____________________________________________________________________
max_pooling1d_2 (MaxPooling1D)   (None, 49, 128)   0
____________________________________________________________________
dropout_1 (Dropout)          (None, 49, 128)       0
____________________________________________________________________
dropout_2 (Dropout)          (None, 49, 128)       0
____________________________________________________________________
flatten_1 (Flatten)          (None, 6272)          0
____________________________________________________________________
flatten_2 (Flatten)          (None, 6272)          0
____________________________________________________________________
merge_1 (Merge)              (None, 6272)          0
____________________________________________________________________
dense_1 (Dense)              (None, 2)             12546
====================================================================
Total params: 513,202.0
Trainable params: 513,202.0
Non-trainable params: 0.0
```

Model summary of the Siamese CNN model

Observations of several CNN models with varying dropout and number of layers

Model	Number Of layers	Dropout	Validation Loss	Training Loss
CNN	2	0.5	0.54606734	0.544829
CNN	3	0.4	0.6147397	0.61473978

After training the model using pre-trained gloVe embeddings, the observations (in the table above) can be made. It is observed that when 2 layers are used, the validation loss is lower than when three layers are used. This is because unlike how CNNs work in text, most of the learning happens in the first two layers itself as there are fewer features to be learnt. The models above have run for a total of 10 epochs.

Siamese Architecture with Long Short-Term Memory Networks:

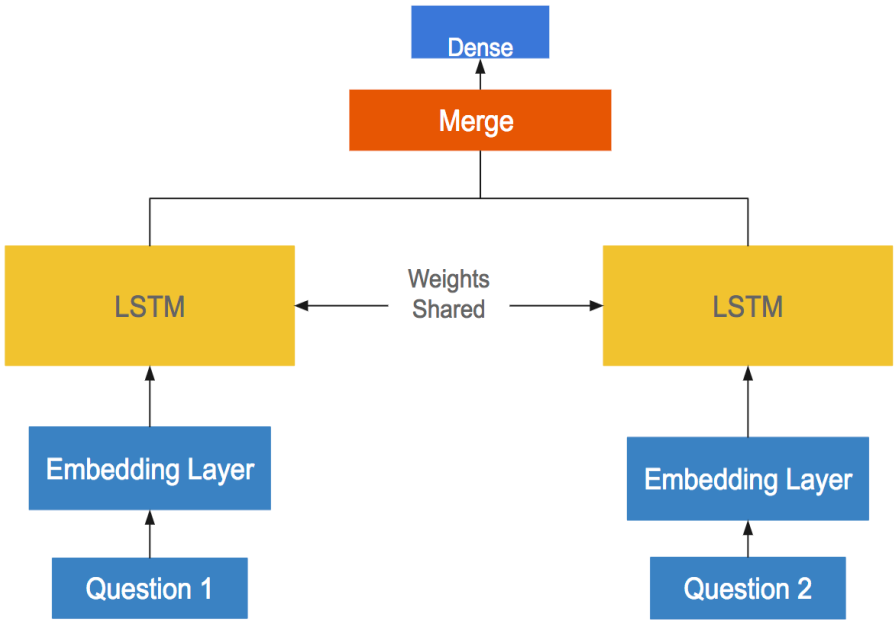
The Model is designed using Keras Deep Learning library. The layers of the model are stacked one after another Sequentially. Question1 and Question2 are given has input to the first layer and similarity measure is produced as output from the last year.

Embedding Layer (First Layer): This layer is the first layer in Deep NLP Architectures. This layer maps a given text input, each word to its corresponding Word Embedding. The LSTM networks works on the vector representation of the words and not the word itself as explained before. We limited the number of words, which the embedding layer can detect to 200,000 most common words in the training data. We also need to normalize (standardize) the length of each question, we set the max number of words in a question to 30 words. If a question is smaller than 30 words, we append zero vector to make it to 30 words.

LSTM layer: This is a simple LSTM network, which takes words in a question has input sequentially (words converted to "word embedding" from the Embedding Layer).

Merge Layer: Merge Large is used to combine different vector outputs from the LSTM. Both LSTM's will produce a sentence vector, merge layer combined them to produce a single vector output.

Dense Layer: Converts the vector output from the Merge Layer into a number between 0 and 1, which shows the measure of the similarity predicted by the network.



Siamese Architecture using LSTM

Word2Vec vs Glove Word Embeddings:
Word2Vec and Glove Word Embeddings are widely used for many NLP tasks. We designed a Simple LSTM Siamese model and compared the results of Word2vec vs Glove Embeddings.

Observations:
Word2Vec model has 27million parameters to train compared to 9 million parameters of Glove Vector Representation. This is because of the dimension of word2vec which is 300d compared to glove vectors which are 100d. Word2Vec model takes 1 hour per each epoch whereas glove

vector model was considerably faster taking 20 mins per epoch. However, the log-loss score from both models was almost the same. We used Glove vectors for future models, due to their faster processing time and almost comparable performance to Word2vec vectors.

model - Siamese Networks using simple LSTM (Glove)

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 30)	0
input_2 (InputLayer)	(None, 30)	0
embedding_1 (Embedding)	(None, 30, 100)	9027900
lstm_1 (LSTM)	(None, 100)	80400
lstm_2 (LSTM)	(None, 100)	80400
merge_1 (Merge)	(None, 100)	0
dense_1 (Dense)	(None, 2)	202

Total params: 9,188,902.0
Trainable params: 9,188,902.0
Non-trainable params: 0.0

model - Siamese Networks using Simple LSTM (word2vec)

Layer (type)	Output Shape	Param #
input_9 (InputLayer)	(None, 30)	0
input_10 (InputLayer)	(None, 30)	0
embedding_1 (Embedding)	(None, 30, 300)	27083700
lstm_9 (LSTM)	(None, 100)	160400
lstm_10 (LSTM)	(None, 100)	160400
merge_3 (Merge)	(None, 100)	0
dense_3 (Dense)	(None, 2)	202

Total params: 27,404,702.0
Trainable params: 27,404,702.0
Non-trainable params: 0.0

Model Summary of both LSTM Models

Word Representation	Model	Validation loss (20 epochs)	Time-Taken
Word2Vec(300d)	Siamese LSTM	0.423	1 hour (per epoch)
Glove(100d)	Siamese LSTM	0.434	20 mins (per epoch)

Bidirectional LSTM vs LSTM:
Bidirectional LSTM have performed very well for many NLP applications. We compared the performance of Bi-LSTM against the simple LSTM:

Model Parameters:
Length of Each Question: 30 words
Embedding Layer (Max number of Words): 200000
Train: Validation = 9:1
Word Embedding = Glove 100d vectors
Length of LSTM Layer: 100

Observations:

Model	Drop-out	Validation loss (20 epochs)	Time-Taken (per Epoch)
Bidirectional LSTM	Recurrent Dropout=0.2 Dropout=0.2	0.44385	50 mins
LSTM	Recurrent Dropout=0.2 Dropout=0.2	0.43454	20 mins

Bidirectional LSTM suffers from Overfitting. Bi-LSTM stopped in 7 Epochs. The Validation loss started to increase after 5 epochs. This functionality if Implemented using Keras Model Checkpoint and Early Stopping functions. LSTM model also suffered from overfitting (Stopped in 12 Epochs). The ideal solution would be Tuning Recurrent Dropout and Dropout for Bidirectional LSTM. We did not Implement it because of Time Constraints. Each epoch of Bi-directional LSTM approximately an hour.

NOTE: We finalized on LSTM + Glove Vectors for further analysis, due to comparative performance and faster implementation observed from the above analysis.

Dropout:
Dropout is one of the most important regularization techniques for Deep Neural Networks. Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass. It helped reduced overfitting and improve performance on unseen data for many applications.

Dropout analysis on Siamese LSTM Network:

Model	Drop-out	Validation loss	Training Loss
LSTM	Recurrent Dropout=0.2 Dropout=0.2	0.4345	0.4301
LSTM	Recurrent Dropout=0.4 Dropout=0.4	0.4532	0.4476
LSTM	Recurrent Dropout=0.0 Dropout=0.0	0.4632	0.4375

Dropout of 0.2,0.2 provides the best generalization. LSTM with dropout of 0.4,0.4 suffers from reduced performance on both train and validation set. LSTM without dropout was also implemented, as expected it suffers from overfitting. We could have also tried randomized dropout (we did not implement it due to time complexity)

Steps to Improve Performance:

Text Preprocessing:

All the analysis prior to this point, used minor text processing. We implemented Deep LSTM models on text with almost no preprocessing. We wanted to try different preprocessing steps and see how they would affect the performance.

Digit Replacement:

Word2vec and glove vectors don't have representation of digits directly. They have to be replaced by 'n' before sending the text through the embedding layer of Siamese architecture.

Handling Special Words:

Words like I'd, I'm cause problems when tokenized using standard tokenizers.
Example: I'd - I + d and I'm - I + m.
When they are split into separate parts, the meaning is lost. If we replace the ' character and convert I'd to Id, the word is present in glove vectors but the word vector is not closely related to "I would". So, these words were handled separately.
Example: I'd - I would, I'm - I am

Stop Words Removal

Stop words increase complexity of system, they might or might not provide any additional information. We compared model with stop words vs without stop words.

Preprocessing Step	Original Text	Modified Text
Number Replacement	I have 100 apples.	I have n Apples
Special Words	I'd run a marathon.	I would run a marathon.
Stop word removal	I have 100 apples.	I 100 apples

Different Text Preprocessing Steps

Imbalance Data Classes:

Training Data and Testing Data might have different distribution of positive and negative examples. In practice, if training data has considerably more positive examples, the model tends towards positive while prediction and vice versa. In our data, Train Set contains 36.92% positive examples and Test set contains 17.46% positive examples. There is clear imbalance in class ratio between train and test sets. We need map the share of positive examples to be the same in Test and Train sets. The weight of one positive example in the train set counts for 0.472 (0.1746 / 0.3692) positive entities in the test set. Similarly, the weight of negative example in the train set is (1 - 0.1746) / (1 - 0.3692) = 1.309. Now the Log loss function becomes

$$\text{Log loss} = -(0.472001959 * y * \log y\_ + 1.309028344 * (1.0 - y) * \log (1.0 - y\_))$$

Where y_ is prediction, y is label

Keras provides a very simple way to provide class weight has a parameter while training.

Note: This is a work around to achieve better performance on Test set. As in this case, the distribution is totally skewed in the Test Set. Ideally, we need to split the Train, Test and validation set, maintaining class balance.

Question Pair Symmetry:
We tried to interchange Question1 and Question2 to see if it effects the model. Ideally model performance should not change if the questions are interchanged. Deep Learning models particularly LSTM's suffer from over fitting. We wanted to check if the model is learning any unwanted features. To perform this analysis, we flipped half of question1 as question2.

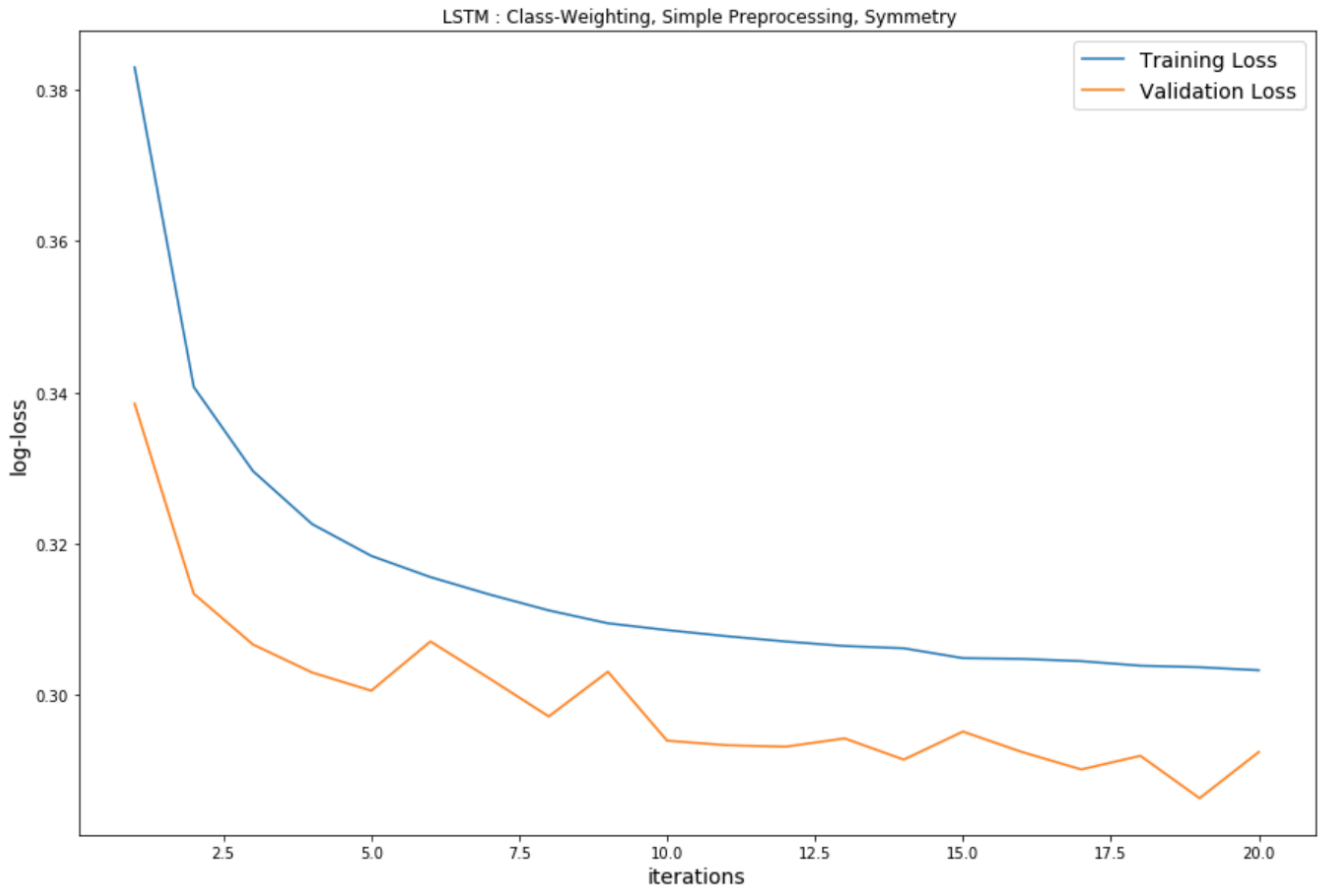
Analysis of Different Steps:

Model Parameters:
Length of Each Question: 30 words
Embedding Layer (Max number of Words): 200000
Train: Validation = 9:1
Word Embedding = Glove 100d vectors
Length of LSTM Layer: 100

Results:

Steps	Train Loss	Validation loss	Public Test loss	Private Test Loss
Symmetry-No , Simple Text Preprocessing -Yes , Class Weighting - No	0.4034	0.4056	0.4041	0.4065
Symmetry-No , Simple Text Preprocessing -Yes , Class Weighting - Yes	0.3045	0.2931	0.3104	0.3144
Symmetry-Yes Simple Text Preprocessing -Yes Class Weighting - Yes	0.3033	0.2925	0.3079	0.3109
Symmetry-Yes , Simple Text Preprocessing -Yes , Class Weighting - Yes, Stop word Removal - Yes	0.3512	0.3567	0.3626	0.3617

Log-loss Comparison table



Training and Validation Loss vs Iteration (for Best Model)

Observations:

1. Some models suffer from under fitting, clearly visible from validation loss lesser than train loss. We feel this is because of the restriction on number of epochs to 20. Training loss was still decreasing at 20 epochs. We did not increase the epochs and try retraining due to time complexity.
2. Secondly, we can observe that Stop Words Removal decreased the performance. This clearly shows that stop words had contextual information which was important very important.
3. Class Weighting has improved the performance drastically. We were surprised by how it improved even the training loss and validation loss.

Summary and Conclusion:

LSTM Models with Dropout-0.4 and Recurrent Dropout-0.4, Preprocessing Steps- Class Weighting, Digit Replacement, Handling Special Words produced the best Log-loss of 0.3079 on Test Set.

References:

1. Contextual Bidirectional Long Short-Term Memory Recurrent Neural Network Language Models: A Generative Approach to Sentiment Analysis - http://www.aclweb.org/anthology/E17-1096
2. A Hierarchical Neural Autoencoder for Paragraphs and Documents - https://arxiv.org/pdf/1506.01057v2.pdf
3. Siamese Recurrent Architectures for Learning Sentence Similarity - http://www.mit.edu/~jonasm/info/MuellerThyagarajan_AAAI16.pdf
4. Efficient Estimation of Word Representations in Vector Space - https://arxiv.org/pdf/1301.3781.pdf
5. GloVe: Global Vectors for Word Representation - https://nlp.stanford.edu/pubs/glove.pdf
6. HDLTex: Hierarchical Deep Learning for Text Classification - https://arxiv.org/pdf/1709.08267.pdf
7. Signature Verification using a "Siamese" Time Delay Neural Network - https://papers.nips.cc/paper/769-signature-verification-using-a-siamese-time-delay-neural-network.pdf
8. Dropout: A Simple Way to Prevent Neural Networks from Overfitting - http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf
9. Understanding CNNs for NLP - http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/
10. Comparison study of CNNs and RNNs for NLP - https://arxiv.org/pdf/1702.01923.pdf
11. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
12. Term Frequency Inverse Document Frequency - http://www.tfidf.com/