1 Introduction

Quora is a popular question-answer platform where users can create, edit, and comment on answers and questions. One challenge Quora faces is dealing with repeated questions and answers, which is a difficult task. Natural language processing (NLP) is an effective method for determining duplicate or similar questions. Misclassifying duplicate questions can be costly. Identifying semantically identical questions on Q&A platforms like Quora is crucial for maintaining high-quality content and enhancing the overall user experience. Detecting duplicate questions is challenging because natural language is highly expressive, and the same intent can be conveyed using different words, phrases, and sentence structures. Machine learning and deep learning methods have shown superior results over traditional NLP techniques in identifying similar texts.

2 Theory

2.1 Neural Network

Neural networks are inspired by biological neural networks (the human brain). They are computing systems with interconnected nodes called neurons. Using algorithms, neural networks can recognize hidden patterns and correlations in raw data, cluster and classify it, and continuously learn and improve.

A neural network with more than one hidden layer is generally referred to as a Deep Neural Network (DNN). Neural networks began as an attempt to mimic the architecture of the human brain to perform tasks that conventional algorithms had little success with. They are more accurately referred to as artificial neural networks (ANNs).

Neural networks are a means of performing machine learning, in which a computer learns to perform tasks by analyzing training examples. Usually, these examples have been hand-labeled in advance. For instance, an object recognition system might be fed thousands of labeled images of cars, houses, coffee cups, and so on, and it would find visual patterns in the images that consistently correlate with particular labels.

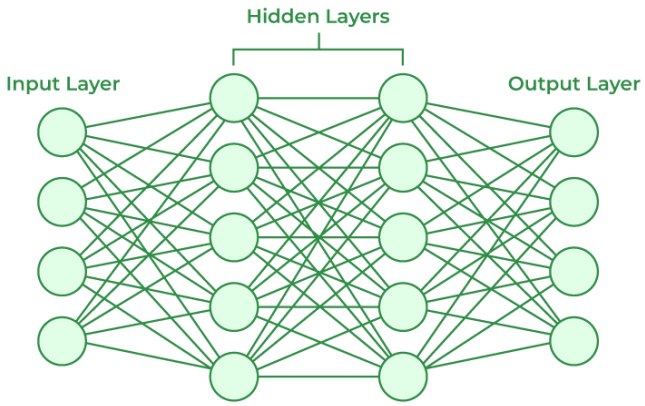


Figure 1: Simple Neural Network Architecture

2.2 RNN

Recurrent Neural Networks (RNNs) are a crucial variant of neural networks extensively used in Natural Language Processing (NLP). An RNN is a type of neural network where the output from the previous step is fed as input to the current step.

In traditional neural networks, all the inputs and outputs are independent of each other. However, for tasks like predicting the next word in a sentence, the previous words are essential, necessitating the ability to remember them. RNNs address this issue with the help of a hidden layer. The main and most important feature of RNNs is the hidden state, which retains information about a sequence.

RNNs are called recurrent because they perform the same task for every element of a sequence, with the output depending on the previous computations. They have proven to be highly successful in NLP, especially with their variant Long Short-Term Memory (LSTM) networks, which can remember information for longer periods than traditional RNNs.

2.3 LSTM

Long Short-Term Memory (LSTM) networks were designed to handle long-term dependencies. The key idea that sets LSTMs apart from other neural networks is their ability to remember information for extended periods without the need for repeated learning, making the process simpler and faster. This type of recurrent neural network includes an inbuilt memory system for storing information, enabling it to retain and utilize past information effectively.

Advantages of LSTM Networks:

1. **Handling Long-Term Dependencies:** LSTMs can remember information for long periods, making them suitable for tasks where context and sequence are important.

2. **Mitigating Vanishing Gradient Problem:** LSTMs address the vanishing gradient problem common in traditional RNNs, allowing them to learn and retain long-term dependencies more effectively.

3. **Improved Accuracy**: By retaining important information over time, LSTMs improve the accuracy of models in tasks such as language modeling, machine translation, and speech recognition.

4. **Versatility:** LSTMs are versatile and can be applied to various tasks, including time series prediction, handwriting recognition, and anomaly detection.

5. **Efficiency:** LSTMs can reduce the need for extensive data preprocessing and feature engineering, streamlining the development process for complex models.

These advantages make LSTM networks a powerful tool for many applications, particularly those involving sequential data and complex patterns.

Note: In this Experiment this model is used

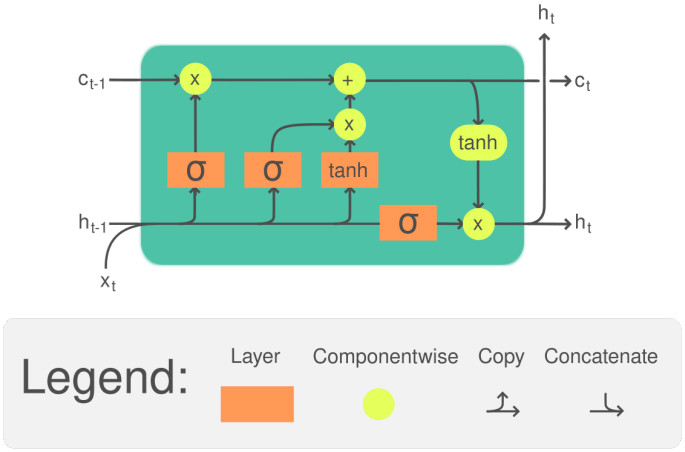


Figure 2: LSTM Architecture

3 Packages

3.1 Keras

Keras is an open-source neural-network library written in Python. It can run on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, Keras focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System) by François Chollet, a Google engineer.

Keras includes numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to simplify working with image and text data. This makes coding deep neural network models easier and more efficient.

In addition to standard neural networks, Keras supports convolutional and recurrent neural networks. It also supports other common utility layers like dropout, batch normalization, and pooling, making it a versatile tool for developing a wide range of neural network models. The following Keras components were used in these experiments:

- `keras.preprocessing.sequence.pad_sequences()`
- `keras.preprocessing.text.Tokenizer()`
- `keras.optimizers.Adam()`
- `keras.layers`
- `keras.models`

3.2 Tensorflow

TensorFlow is a free and open-source software library for data ow and di erentiable programming across a range of tasks. It is a symbolic math library, and is also used for

machine learning applications such as neural networks. It is used for both research and production at Google. TensorFlow can run on multiple CPUs and GPUs.

4 Word Embedding

A word embedding is a learned representation for text where words with similar meanings have similar representations. This approach to representing words and documents is considered one of the key breakthroughs in deep learning for addressing challenging natural language processing (NLP) problems.

Word embeddings are a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to a vector, and the vector values are learned in a manner akin to neural networks, which is why this technique is often associated with deep learning.

The core idea is to use a dense, distributed representation for each word. Each word is represented by a real-valued vector, typically comprising tens or hundreds of dimensions. This contrasts with sparse word representations, such as one-hot encoding, which require thousands or millions of dimensions.

4.1 Embedding Layer

An embedding layer, aptly named for its function, is a word embedding learned jointly with a neural network model for specific natural language processing tasks, such as language modeling or document classification.

This process necessitates that document text be cleaned and prepared so that each word is one-hot encoded. The size of the vector space is specified as part of the model-commonly 50, 100, or 300 dimensions. Initially, the vectors are filled with small random values. The embedding layer is placed at the front end of the neural network and trained in a supervised manner using the backpropagation algorithm.

In this setup, one-hot encoded words are mapped to word vectors. If a multilayer perceptron (MLP) model is used, the word vectors are concatenated before being input to the model. In the case of a recurrent neural network (RNN), each word is processed as an individual input in a sequence.

Although learning an embedding layer requires a substantial amount of training data and can be slow, it effectively learns an embedding that is both targeted to the specific text data and aligned with the NLP task at hand.

4.2 Word2Vec

Word2Vec is a statistical method for efficiently learning standalone word embeddings from a text corpus. Developed by Tomas Mikolov and colleagues at Google in 2013, Word2Vec was created to enhance the efficiency of neural-network-based training for word embeddings and has since become the de facto standard for pre-trained word embeddings.

The Word2Vec approach involves analyzing the learned vectors and exploring vector math on word representations. For instance, subtracting the vector for "man" from "king" and adding the vector for "woman" results in a vector close to "queen," capturing the analogy that "king" is to "queen" as "man" is to "woman."

These word embeddings effectively capture syntactic and semantic regularities in language. Each relationship between words is characterized by a relation-specific vector

offset, allowing for vector-oriented reasoning based on these offsets. For example, the male/female relationship is automatically learned, and with these vector representations, "King" - "Man" + "Woman" results in a vector very close to "Queen."

Two different learning models are part of the Word2Vec approach:

- **Continuous Bag-of-Words (CBOW) Model:** This model learns the embedding by predicting the current word based on its context.
- **Continuous Skip-Gram Model:** This model learns by predicting the surrounding words given a current word.

Both models focus on learning about words based on their local usage context, defined by a sliding window of neighboring words. The size of this window is a configurable parameter that has a significant impact on the resulting vector similarities. Larger windows tend to produce more topical similarities, while smaller windows capture more functional and syntactic similarities.

The key benefit of Word2Vec is its ability to learn high-quality word embeddings efficiently, with low space and time complexity. This allows for the creation of larger embeddings (with more dimensions) from much larger corpora of text, encompassing billions of words.

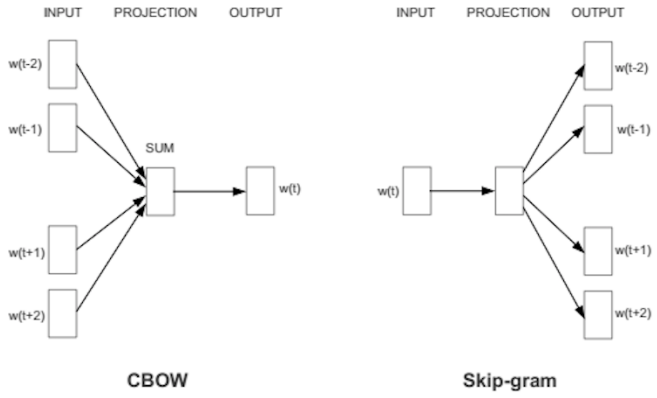


Figure 3: word2vec Architecture

5 Data Description

For this project, we utilized the publicly available Quora Question Pairs dataset from Kaggle. The dataset used for our experiment is the training set provided on the Kaggle website. This training dataset, intended for the competition, contains 404,290 question pairs and has a size of 60.4 MB.Each instance in the dataset includes the ID of the question pair, unique IDs for both questions, the full text of each question, and a target variable indicating whether the two questions are duplicates (i.e., have the same meaning)

Columns	Description
id	unique value for the question pair
qid1	unique value for the first question pair
qid2	unique value for the second question pair
question1	full text of the first question
question2	full text of the second question
is_duplicate	If question pair is duplicate then 1, otherwise 0

Table 1: Description

The dataset includes question pairs that span a wide range of topics, including education, technology, sports, philosophy, politics, culture, economics, social issues, entertainment, and more. Additionally, some questions feature special characters, such as mathematical symbols and foreign language characters.

We have divided the data into training and validation sets with a 70:20 ratio. The training set contains 323,429 instances, while the validation set has 80,858 instances.

6 Implementation

6.1 Dataset Preprocessing

Preprocessing starts with extracting text for question 1 and question 2 from the file, along with the label (1 for duplicate, 0 for non-duplicate). There are various preprocessing methods available before preparing the text for modeling.

Our preprocessing approach began with dataset cleansing, which involved removing special characters (such as `! # $ % & () + ,- / : ; <●>? @ [/]`) from the `question1` and `question2` columns, as they do not contribute meaningfully to the context of a sentence. For instance, an analysis of the Quora dataset revealed that 99.87% of questions contain the question mark symbol, while other symbols like the full stop appear in 6.39% of questions, and fewer than 1% contain mathematical or programming symbols such as `, =`, etc.

We noted that removing stop words could make the relationships between duplicates, as illustrated in Table, less clear. Therefore, we decided against removing stop words, as this could alter the meaning of the sentences.

Regarding lemmatization and stemming: Lemmatization returns a word to its base or dictionary form, while stemming removes word suffixes. To preserve as much of the original meaning of the sentences as possible, we chose not to apply either of these techniques to our datasets.

S.No.	Original	Stop words removed
Q1	Why do people not like Donald Trump running for president?	Why, people, like, Donald, Trump, running, president, ?
Q2	Why are people so against Donald Trump running for president?	Why, people, Donald, Trump, running, president, ?

Table 2: Examples of sentences losing initial meaning after stop word removal.

Then we converted all the texts to lowercase. After the cleaning step, each sentence undergoes a tokenization process, where it is split into individual words based on blank spaces. We created a dictionary (also known as a word index) of 89,983 unique words. In this dictionary (also called a corpus), each unique word is assigned an index. This dictionary is used to replace every word in a sentence with its corresponding index, resulting in word vectors. For example, the question:

> **Input:** What is the step by step guide to invest in share market in India?
> **Output:** [2, 3, 1, 1222, 59, 1222, 2566, 7, 579, 8, 763, 384, 8, 36]

Next, we encode the sentences using the `text_to_sequences()` function. It is important to encode the two columns of questions in the dataset separately.

The embedding layer requires that all vectors have the same length, which is not the case initially as sentences naturally vary in length. To address this, we apply padding to both columns separately with a maximum word size of 100. This converts all questions of variable length into a fixed length.

For instance:

Input: Whats the best online calendar/scheduling/booking system? Something that allows me to show my availability so customers can request to meet me. When I accept their request, the appointment pops into our calendars and reminders are automated.
Output: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 26, 1, 19, 20, 784, 22234, 9721, 3068, 211, 404, 29, 3665, 62, 7, 442, 17, 11647, 2965, 13, 1967, 7, 820, 62, 39, 4, 1300, 79, 1967, 1, 4475, 14553, 108, 214, 15925, 17649, 11, 6685]

The main purpose of these steps is to remove noise, i.e., elements that do not add meaning to the sentences.

Word2Vec
To create the embedding matrix for the embedding layer, we utilized a Word2Vec model trained on our dataset. The Word2Vec model converts words into dense vectors of fixed size, capturing semantic relationships between them.

Specifically, we trained the Word2Vec model on our data to learn word embeddings that represent the context and meaning of words in our particular domain. The resulting word vectors are then used to populate the embedding matrix.

The embedding matrix is a crucial component of the embedding layer in our model. It maps each word index to its corresponding word vector, allowing the model to leverage pre-learned semantic relationships between words during training. This process enhances the model's ability to understand and generate meaningful patterns based on the context provided by our dataset.

6.2 Implementation of LSTM with Common Words Feature

In this neural network architecture, we use a pair of questions as the two inputs.

The first hidden layer of this network is an `Embedding` layer, which represents each word as a dense vector using word embeddings. We specify three arguments to the `Embedding` function: the input dimension, output dimension, and the input length, which are set to `vocab_size+1`, 300, and 100, respectively.

The architecture consists of the `Embedding` layer followed by separate `LSTM` layers applied to each of the question inputs in parallel.

The outputs of the `LSTM` layers and the Common Words Feature are processed through separate `Dense` layers in parallel, as illustrated in the architecture diagram.

Subsequently, the outputs from the `Dense` layers are merged using a `Concatenated` layer.

The merged output is then passed through a series of layers: `Batch Normalization` (to speed up and optimize training), `Dropout` (to prevent overfitting), and another `Dense` layer.

The output undergoes another series of `Batch Normalization` and `Dropout`. Finally, the output from the `Dropout` layer is passed through a fully connected `Dense` layer to produce the final classification. A `Sigmoid` activation function is applied at the final output layer.

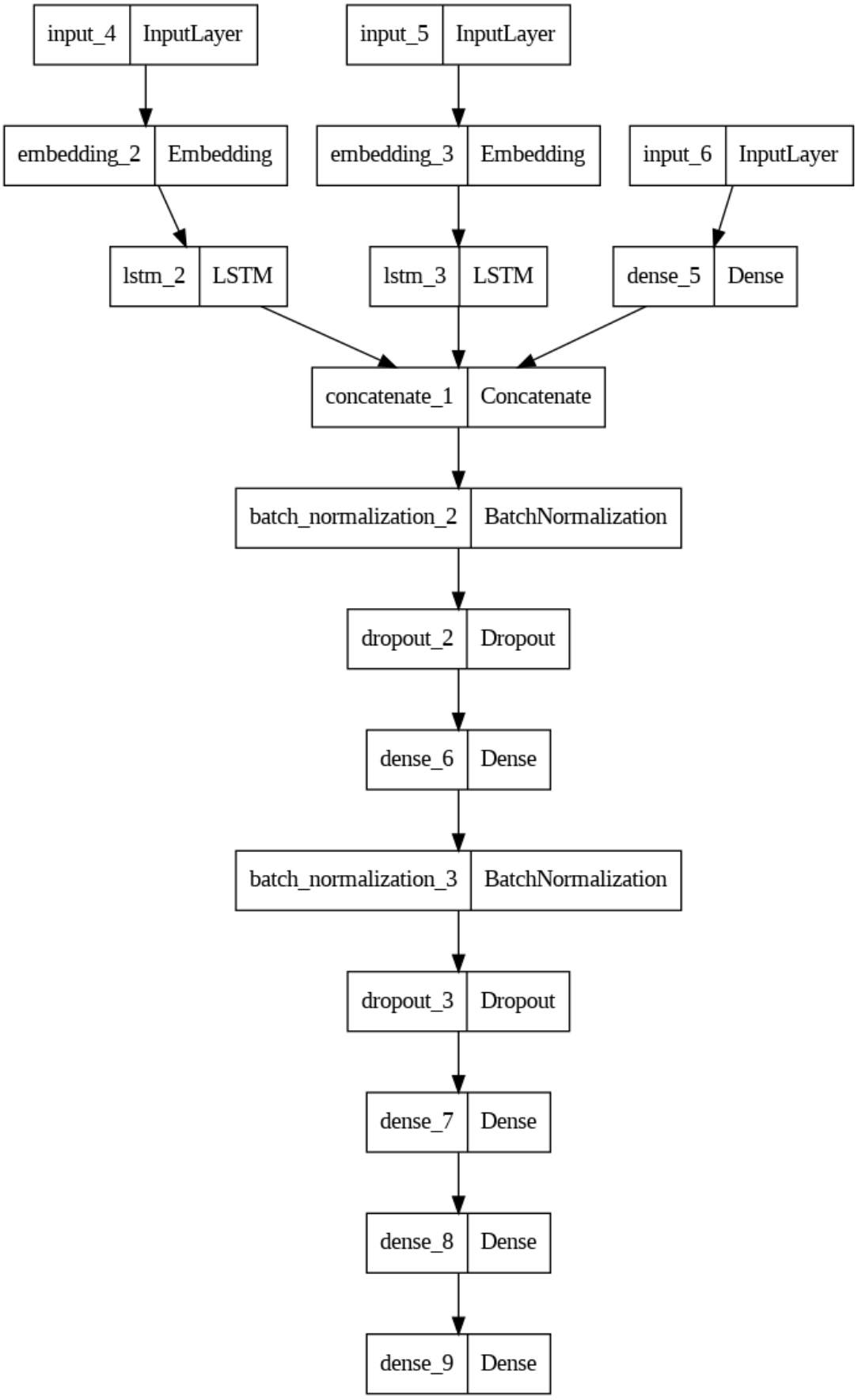


Figure 4: Model

7 Result

Below table shows Accuracy , log loss of the LSTM with Common Words model. We set the batch size to 1024.

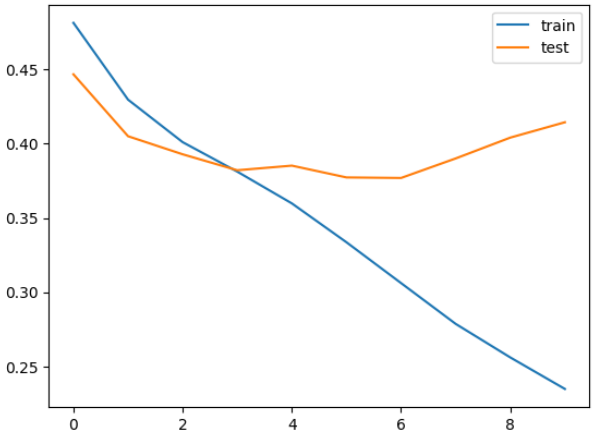
S.No.	Batch Size	Accuracy(%)	Log loss
1	1024	82.21	0.4144

Table 3: Accuracy result for LSTM with Common Words Feature

.

I have plotted the 2 graphs :

- Accuracy v/s Epoch



- Loss v/s Epoch