A Study on Deep Learning Sentimental Analysis on Google Play Store Reviews using Long Short-Term Memory (LSTM)

Author: Koh Kai Chun Samuel

Ngee Ann Polytechnic

Data Science

Author Note

This research project was conducted as part of a school assignment and is intended for portfolio purposes. No external finding was received for this project. Correspondence regarding this report should be addressed to Koh Kai Chun Samuel, School of InfoComm Technology, Diploma in Data Science. Email: samuelkoh17@gmail.com.

Abstract

Sentiment analysis has gained significant importance in recent years, particularly in interpreting user-generated content. This report focuses on applying sentiment analysis to Google Play Store reviews, a domain where understanding user sentiment is crucial for app developers, marketers, and users. With millions of reviews generated daily, extracting meaningful insights from this text-rich data is a challenging task.

The problem statement is to attempt to **apply sentimental analysis** on google play store reviews to **classify** the app review score **with the help of recurrent neural network layers**.

Analyzing Google Play Store reviews presents several complexities. Textual reviews often contain linguistic nuances, sarcasm, and a wide range of expression styles, making it challenging to accurately quantify user sentiment. Traditional methods of review analysis may fall short in capturing the subtleties of user emotions, hence necessitating the development of advanced machine learning models.

The primary objective of this project is to build a robust sentiment analysis model capable of predicting review scores for applications within the Google Play Store reviews accurately. Achieving this goal holds immense significance as it can benefit app developers in improving their products, aid marketers in understanding user feedback, and assist users in making informed decisions when choosing apps. Success in this project entails the development of a model that outperforms traditional methods and provides valuable insights into user sentiment.

In our project workflow, we commence with the Data Loading and Processing phase, where each student selects a specific Google Play Store app. They are responsible for collecting review data and extracting the essential content and score columns. Data preprocessing, which includes cleansing and numerical encoding, is a collaborative effort performed as a group.

Moving on to the Developing Sentiment Analysis Models stage, we independently construct at least one model. These models encompass word embedding layers, which can utilize pre-trained word embeddings or be trained on the dataset itself, alongside RNN layers like LSTM or GRU. The model development process adheres to a systematic approach, involving the creation of a baseline model, scalability assessments, regularization techniques, and hyperparameter tuning. Throughout this phase, we continuously monitor and analyse model performance during training.

Following model development, we proceed to the Model Evaluation step. Here, we employ testing data, represented as X\_test and y\_test, to rigorously assess the models' performance. Comparative analysis and discussions ensue, ultimately leading to the recommendation of the best-performing model.

Subsequently, the chosen best model is utilized in the Making Predictions phase. It is employed to predict sentiment based on user input, where numeric tensors are generated from this input and then input into the model to produce sentiment scores. This comprehensive approach ensures that our sentiment analysis system is not only well-constructed but also accurately predicts user sentiment.

In conclusion, this project aims to advance sentiment analysis techniques in the context of Google Play Store reviews. By tackling the complexities of understanding user sentiment from textual data, we hope to provide valuable insights to app developers, marketers, and users. The journey through this project also opens doors to future applications of the developed model in various domains.

We first do one-hot encoding in the y train and y test data. One hot encoding refers to each tokenized word or token, into a binary vector where only an element is hot (set to 1), and the rest, not present as cold (set to 0). Creating a Baseline Model, the general workflow for model tuning is to:

- Tune the layer for embedding units

- Number of hidden layers

- Learning Rates

- Batch Sizes

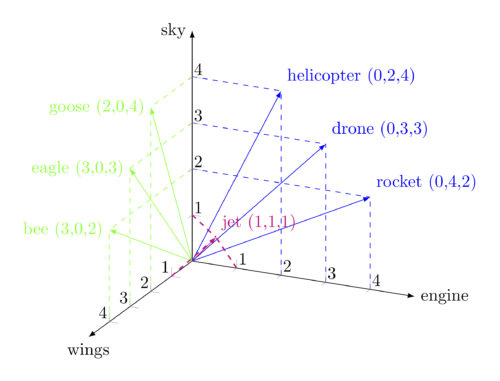
- Number of nodes in each hidden layer

- Dropout layers

### 3.3.1. Tuning Embedding Layers

We discuss how word embeddings can help map and capture relationships between data. It is a way to represent words, as vectors, in a high-dimensional space. Its purpose is to capture the relationships between words.

Firstly, we talk about word embeddings in general. Traditional text processing represents words, using word indices in a vocabulary. Then, embedding layers transform these discrete word representations, into the continuous vectors.



Then, in a vector space, we first discuss the X and Y, and Z axis. Since each dimension represents an aspect of a feature. In this example, since sky, engine and wings represent three different features, hence the word is mapped according to its positions.

Figure 11 - Illustration of word vectors representing in a dimensional space.

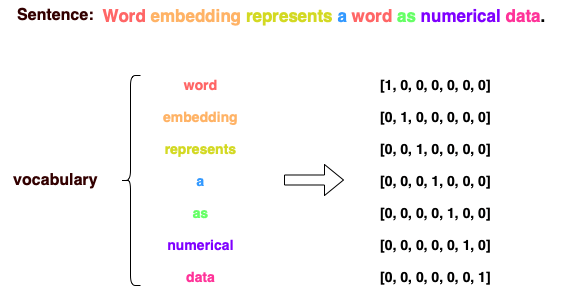
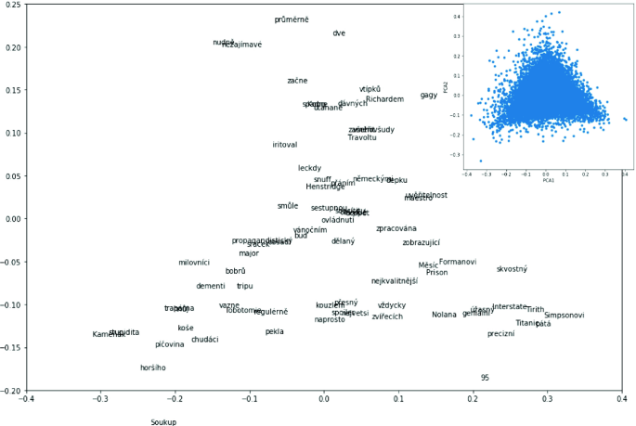


Figure 12 - Illustration of mapping of word representations to numerical data.

According to Baeldung, these as coordinates will be mapped into numerical data, and hence embedding layers use dimensions like these to capture patterns in data.



We may consider using dimensionality reduction methods, like Principal Component Analysis (PCA), to sum up its dimensions into less dimensions, while retaining the most important information. It can also reduce computational resources. However, since PCA may discard dimensions that have lower variance, this may result in a loss of fine-grained information, present in the original high-dimensional space. Hence, by using embedding layers of more dimensions, we capture this semantic nuance more easily.

Figure 13 - Using 2 Principal Components (PCs) to capture word relationships may lose granularity.

According to Baeldung’s documentation, a dataset with less than 100,000 sentences benefits from a lower dimensional embedding. This can be 50 to 100 dimensions. A dataset with larger than that, benefits from a higher-dimensional embedding (200-300 dimensions). Hence, we decided to choose only to experiment 50 to 100 dimensions, since it is for smaller dataset with only about 30,000 rows, or sentences. Then, we compare this custom word embedding, to a pretrained word embedding, Global Vectors for Word Representation (GloVe).

The best approach is to experiment, using different dimensionalities, and evaluating the performance of the model, on validation set. We also start off with a simple RNN model with 32 nodes on one layer.

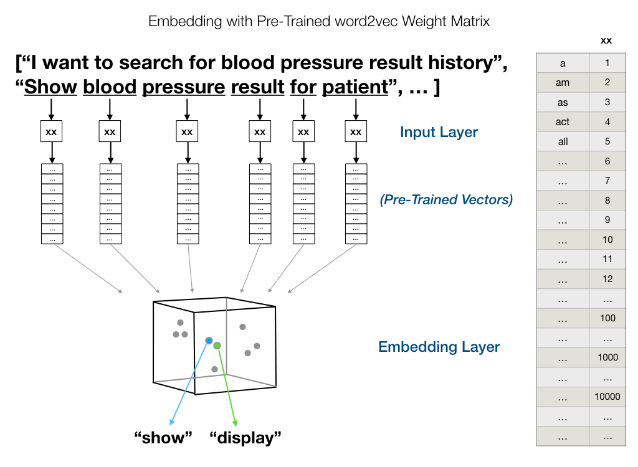
Hence, tuning the embedding layer values are: 60, 70, 80, 90, and 100.

A group of graphs with different colored lines

Description automatically generated

Figure 14 - a study of increasing dimensionality of custom word embedding layer (60, 70, 80, 90, 100), with respect to accuracies.

Using iterations of 60, 70, 80, 90 and 100, 90 dimensions has the highest validation accuracy, about 53%. Although all models overfitted, and there is an obvious upward trend for validation loss, we realize embedding layer = 100 seems to overfit more, with lowest validation accuracy at 50%. Hence, 90 dimensions for custom embedding layer is most suitable. More dimensions mean more details of relationships are captured, that may lead to more noise in the model. The noise did not allow model to generalize new unseen data but learn too well with training patterns.

We define a pretrained embedding matrix, GloVe. After initializing the weights of the matrix of embedding layer in the neural network model, it provides an initial representation of words for input text.

According to Manigiavacchi, Microsoft Principal Data Scientist, he suggests that embedding layers of GloVe maps each word into a vector representation, automatically learnt before. GloVe allows a more powerful representation of data. Also, we set trainable of GloVe weights, to true, to allow the layer to suit the needs of our sentimental analysis of Netflix. This is to learn values of Netflix context specifically.

Figure 15 - Illustration of GloVe's algorithm in representing words in dimensional space.

**Comparison between pretrained GloVe embeddings and custom embeddings (90 dimensions)**

A group of graphs showing different colored lines

Description automatically generated with medium confidence

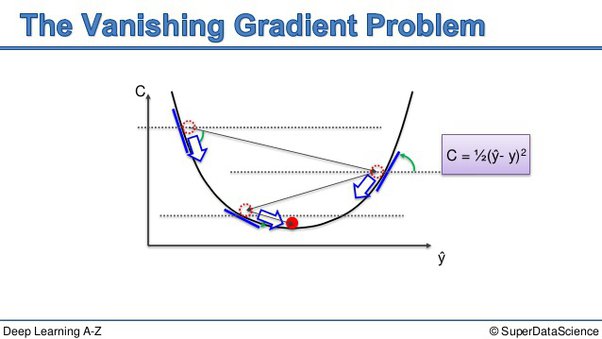
Figure 1016 - A study of comparison of GloVe (pretrained word embeddings), with custom embedding layer.

Lastly, the comparison between GloVe and best performing custom embedding layer: GloVe performed better in validation accuracy by 1% improvements, although the upward graph of loss still shows overfitting. Since GloVe has seen more patterns of words before, it can predict the right rating given a review more accurately. Hence, GloVe with 50 dimensions is used.

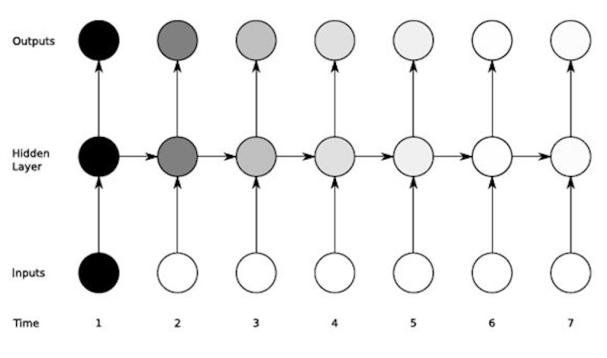
GloVe with 100 dimensions captures more noise, and hence is unable to generalize the unseen reviews.

### 3.3.2. Addressing Vanishing Gradient Problem

We saw that the weight update in section 3.3.1 has problems and is unable to update weights to push for better validation accuracy. In this section, Long Short-Term Memory (LSTM) models are introduced to overcome the weight update problem, also known as vanishing gradient problem.



According to book Deep Learning Pipeline, going down the longer sequences, the gradients get smaller resulting in weights to never change at lower layers. The gradients coming from longer sequences must go through continuous matrix multiplications, due to chain rule. As they approach smaller sequences, if they have small values, say <1, they shrink exponentially till they vanish. This makes it impossible for model to learn.



LSTM networks, an advanced version of RNNs, can remember long-term dependencies. Designed to remember information for long periods of time, without dealing with vanishing gradient problem. LSTMs have a similar chain-like structure, just like RNNs.

However, the repeating module has a different structure, instead of having single neural network layer (tanh), there are 4.

Figure 17 - Given C is the loss function, its derivative reaches a local minimum, hence it no longer updates loss.

Figure 18 - Lighter gradients represent vanishing gradients in RNN model.

### 3.3.3. Number of Hidden Layers

A graph of different colored lines

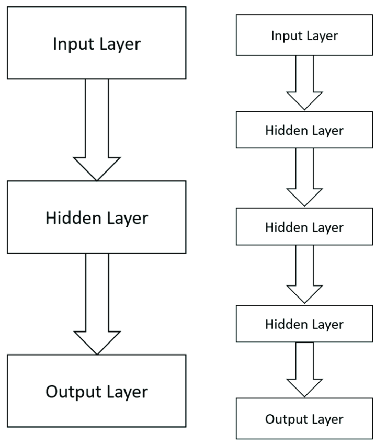
Description automatically generated with medium confidence

Figure 19 - A study of increasing number of hidden layers with respect to model metrics.

Deeper LSTM architectures can potentially capture more hierarchical features in the input data. For sentiment analysis, it means the model may learn to recognize sentiment related patterns, at different levels of abstraction.

Deeper models have higher capacity of learning complex patterns, and can be advantageous in sentiment analysis tasks, like dealing with diverse language expressions, or nuanced sentiments with varying sentence structures.

Figure 20 - Comparison of shallow vs deep networks LSTM.

The approach was to scale up the model gradually, in model architecture, using an increasing number of hidden layers. This makes the model deeper, and it potentially mitigates overfitting by scaling up the model. Models of 1, 2, 3, and 4 hidden layers were tested, and we chose 4 layers as the best model.

We notice that the architecture with 2 hidden layers, each containing 32 nodes, was chosen. This was with the goal of providing the model with sufficiently complex model, and capacity to learn more intricate patterns and representation in the data. It has the lowest loss, although all models have overfitted (upward trend of increasing loss). Although 4 hidden layers had a 53% accuracy, its train accuracy was slightly lowered, hence mitigating overfitting. The gap between training and validation accuracy is reduced, then we can use other methods to increase validation accuracy. Using reduction of loss of 2.4 – 1.8 = 0.6 loss, compared to 1 hidden layer loss, we conclude that lower losses still help better (it was close to predicting 4-star reviews as 5-star reviews, which we discuss later in confusion matrix). Although that was a loss in accuracy, its closeness to prediction is still more considered.

This layered architecture allows a hierarchical representation of features and patterns in the data. Each layer tries to learn increasingly abstract and higher-level representations, based on input from its previous layer.

Multiple hidden layers enable the model to capture non-linear relationships and interactions in data. Hence, it allows more nuanced and comprehensive understanding of reviews sentiment patterns.

### 3.3.4. Tuning Learning Rate

Learning rate is a hyperparameter that determines the size of steps taken, during optimization, like gradient descent. This is when updating the weights of the neural network. It controls the magnitude of updates to the model’s parameters, in the direction that minimizes the loss function. Since learning rate is a crucial hyperparameter in training neural networks, a higher learning rate leads to faster convergence for models. But it may risk overshooting the optimal solution, or it will oscillate around the local minima.

A graph of a function

Description automatically generated

Figure 21 - Illustration on effects of learning rate with respect to weight updates.

In this section, we evaluate between learning rates of 0.001, 0.002, 0.005, and 0.01. The lower bound represents lower learning rates that are most stable, but upper bound of our value of testing represents higher learning rates that may be not as stable. We likely observe that lower learning rates help in achieving smoother learning rates, but not too slow as it can take a smaller number of epochs to converge the model. This can save us less computational resources and time.

A group of graphs with numbers

Description automatically generated with medium confidence

Figure 22 - A study of learning rate tuning with respect to model metrics (accuracy and loss)

The reasoning behind 0.005 is we observe that medium learning rates help to achieve a smoother convergence (the validation accuracy starts plateauing towards epochs 20 onwards). This results in stable training progress. Although overfitting is still not solved, we still see the highest in validation accuracy (54%, around a 2% improvement as compared to default learning rate at 0.001). In terms of validation loss, it has a slight dip towards epochs 25-30, meaning that the model continues to learn patterns and make close enough predictions, yet they are still wrong.

Hence, we select learning rate = 0.005.

### 3.3.5. Tuning Batch Sizes

Tuning batch sizes is finding the optimal number of samples, to include in each mini batch during training. This batch size is a hyperparameter, determined by the number of data points processed, before updating the model’s parameters. In neural network training, data is divided into batches, where each batch contains a subset of total dataset. Then, batch size specifies the number of samples for each batch. i.e. batch size of 32 means the model will process 32 samples, before weight updates.

In this section, we experiment the size of batch sizes, trying different sizes to evaluate their impact on training dynamics, its convergence speed, and generalization performance.

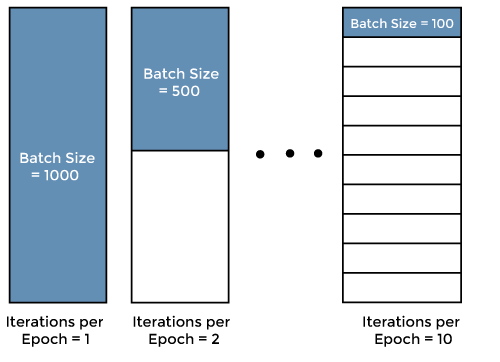


Figure 23 - Illustration of how batch sizes work. The greater the partition of match sizes, the less it evaluates each sample, and the more weight updates of the model are done.

A graph of different colored lines

Description automatically generated with medium confidence

Figure 24 - Batch size tuning of 8, 16, 32, 64, 128 and 256.

From the evaluation, we tested batch size, all to the exponent of 2. This is due to models generally working better with numbers that have this exponent. These sizes are 8, 16, 32, 64, 128, and 256. Batch size = 128 had the highest validation accuracy at around 55%, a 1% improvement on previous results, or comparing with other batch size validation accuracies. The lowest validation accuracy is batch size = 8, and it is unable to push for better accuracy.

Then, larger batch sizes allow more efficient computation, as the model processes more samples in parallel. Since hardware accelerators are optimized for batch processing, with a batch size of 128, it can utilize computational resources. While smaller batch sizes introduce more noise to parameter updates, larger batch sizes help the model to generalize better to unseen data. The averaged gradients computed, over larger batches, can provide a more accurate estimate of the true gradient. This potentially led to a better generalization performance. Using larger batches is less likely to overfit to noisy patterns in data, acting as a regularization to improve the model performance.

Hence, batch size of 128 is used.

**3.3.6.** **Tuning Spatial Dropout 1D**

In normal dropout, we randomly remove nodes from the neural network during training. More precisely, it sets up a probability on each node. Probability refers to the chance that the node is included in the training each iteration of learning. Since a smaller neural network provides less flexibility, hence the chance of overfitting to data is lower. Since randomly removing inputs at each layer in network, the overall network is less sensitive to single outputs. By removing weights from training process at random, dropout will force the other weights to participate in the learning process of that iteration, resulting in final weight values better spread out more.

A black and white crossword puzzle

Description automatically generated

Figure 25 - Illustration of normal dropout (left) vs spatial dropout (right) concept.

In the right part of the image, we see spatial dropout being applied. Unlike regular dropouts, spatial dropout removes entire channels (in this case, entire columns which correspond to specific features across all word embeddings). This is done by setting them to zero. This is shown by the vertical stripes, where entire columns are blacked out. Hence for word embeddings, this means that all words in the sentence would have specific features of their embedding vectors dropped.Hence spatial dropouts remove entire feature dimensions (in word embeddings), rather than individual elements, and is beneficial for the model to not rely on one feature for word embedding.

A group of graphs with numbers and symbols

Description automatically generated with medium confidence

Figure 26 - a study of the effects of spatial dropout proportion (0.2 to 0.5), against model metrics (accuracy and loss)

Higher spatial dropouts lead to reduced overfitting, since it was observed a more lowered train accuracy as the dropout proportion increases. The lowest train accuracy (0.7) and the highest train accuracy, without dropouts (0.8), observed 10% intervals between the tunings. Hence, this reduced overfitting by about 10%. Although spatial dropout of 0 observed the highest validation accuracy, we argue that the overfitting issue is still not resolved, and hence a good dropout is required. Also, the loss for spatial dropout = 0.5 is the lowest, showing that the model almost made accurate predictions.

Hence, we select spatial dropout = 0.5.

### 3.3.7. Tuning Number of Nodes in Each Hidden Layer



**A group of graphs with numbers

Description automatically generated**



Figure 27 - a study of number of nodes for 2 hidden layers, against model metrics.

In this section, we explore the number of nodes in each hidden layer (64, 128, 256). We observe that the number of nodes = 256 had a significantly better validation accuracy. However, we tried comparing these with other variations of 128 nodes, namely having 2 or 3 hidden layers. This is to ease the decision of number of nodes for hidden layers.

A graph of a number of people

Description automatically generated with medium confidence

Figure 28 - a study of effects of number of nodes against model metrics.

We use the layers with highest validation accuracy, being at 128 nodes for two layers of LSTM. This results in models able to generalize unseen data better. Still, sadly the overfitting problem is not solved. With 128 nodes, it makes computation more efficient, and less resources given smaller models. 128 nodes can learn better patterns.

**Conclusion:**

Based on the findings of this study, these hyperparameters are recommended for optimal performance in the sentimental analysis task, conducted on Google Play Store reviews, using Long Short-Term Memory (LSTM) deep learning model.

Using embedding layer dimensions, we utilize GloVe embeddings with 50 dimensions, as the embedding layer. This choice was based on the analysis with different embedding dimensions, with 50 dimensions demonstrating improved validation accuracy, compared to the other dimensions that were tested.

Addressing the vanishing gradient problem, we incorporate Long Short-Term Memory (LSTM) models, to overcome the vanishing gradient problem that is encountered during weight updates. LSTM models are effective in capturing long range dependencies. This is for sequential data which is important for capturing the task information.

As for the number of hidden layers, we configure the model with 2 hidden layers. This architecture strikes a balance between the model’s complexity and its performance, providing sufficient capability to learn more complex data patterns, while we mitigate overfitting.

Setting the learning rate to 0.005, it is to facilitate the smoother convergence and utilization of computational resources, hence leading to higher validation accuracy, compared to other tested learning rates.

We opt for a batch size of 128, and spatial dropout effectively reduced overfitting by randomly dropping entire channels from input data, hence enhancing the model’s ability to generalize to unseen data.

We lastly configure hidden layers with 128 nodes, and this choice of node count can strike a balance between the model’s complexity and its computational efficiency. We enable models to learn more complex patterns, while avoiding excessing computational resources used.

By using these hyperparameters, the sentiment analysis model can achieve an improved performance in accurately predicting sentiments from reviews from Google Play Store. Hence it enhances practical utility and efficacy in the real world.

References

Footnotes

1 For the purposes of this research, the term “overfitting” refers to the phenomenon, where the deep learning model learns to perform well on the training data, but fails to generalize to new, unseen data. It occurs when the model captures noise or random fluctuations in the training data, instead of underlying patterns, resulting in poor performance on the unseen data.