

**School of InfoComm Technology**

**Deep Learning Assignment**

Diploma in CSF / FI / IT

Apr 2022 Semester

**ASSIGNMENT 2**

(40% of DL Module)

4th Jul 2022 – 12th Aug 2022

**Submission Deadline:**

**Presentation: 12th Aug 2022 (Week 17),**

**Report: 12th Aug 2022 (Friday), 11:59PM**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tutorial Group** | CSF02/T02 | **Group** | 4 |
| **Student 1** | Han Xihe (S10222998D) | | |
| **Student 2** | Lim Long Teck (S10221824G) | | |
| **Student 3** | Lee Wee Kang (S10222162) | | |
| **Student 4** | Hasanah Samri (S10222566C) | | |
| **Student 5** | Yong Zi Ren (S10219524J) | | |

**Table of Contents**

[1. Overview 3](#_Toc111240586)

[1.1. Problem 3](#_Toc111240587)

[1.2. Objective 3](#_Toc111240588)

[1.3. Approach 3](#_Toc111240589)

[2. Overview 6](#_Toc111240590)

[2.1. Play Store app chosen, data collection & app evaluation 6](#_Toc111240591)

[2.2. Data cleansing & data conversion to tensors 7](#_Toc111240592)

[2.3. Data sampling & sampling justification 9](#_Toc111240593)

[3. Sentiment Analysis Models 10](#_Toc111240594)

[3.1. Han Xihe 10](#_Toc111240595)

[3.1.1. Model 1 10](#_Toc111240596)

[3.1.2. Model 2 13](#_Toc111240597)

[3.1.3. Model 3 14](#_Toc111240598)

[3.1.4. Model 4 16](#_Toc111240599)

[3.1.5. Model 5 18](#_Toc111240600)

[3.1.6. Model 6 19](#_Toc111240601)

[3.2. Long Teck 20](#_Toc111240602)

[3.2.1. Model 1 21](#_Toc111240603)

[3.2.2. Model 2 22](#_Toc111240604)

[3.2.3. Model 3 23](#_Toc111240605)

[3.2.4. Model 4 24](#_Toc111240606)

[3.2.5 Model 5 25](#_Toc111240607)

[3.2.6 Model 6 26](#_Toc111240608)

[3.2.7 Model 7 27](#_Toc111240609)

[3.2.8 Model 8 28](#_Toc111240610)

[3.3. Hasanah 29](#_Toc111240611)

[3.3.1. Model 1 29](#_Toc111240612)

[3.3.2. Model 2 31](#_Toc111240613)

[3.3.3. Model 3 31](#_Toc111240614)

[3.3.4. Model 4 32](#_Toc111240615)

[3.4. Wee Kang 34](#_Toc111240616)

[3.4.1. Model 1 35](#_Toc111240617)

[3.4.2. Model 2 36](#_Toc111240618)

[3.4.3. Model 3 37](#_Toc111240619)

[3.4.4. Model 4 38](#_Toc111240620)

[3.4.5. Model 5 39](#_Toc111240621)

[3.4.6. Model 6 40](#_Toc111240622)

[3.5. Yong Zi Ren 41](#_Toc111240623)

[3.5.1. Model 1 42](#_Toc111240624)

[3.5.2. Model 2 43](#_Toc111240625)

[3.5.3. Model 3 44](#_Toc111240626)

[3.5.4. Model 4 45](#_Toc111240627)

[4. Model Evaluation using Testing Data 45](#_Toc111240628)

[4.1. Model performance analysis during testing phase 45](#_Toc111240629)

[4.2. Model performance comparison & evaluation 46](#_Toc111240630)

[4.3. Best model & why it is the best 47](#_Toc111240631)

[5. Prediction with the Best Model 47](#_Toc111240632)

[5.1. How the model can be applied to real-life text inputs 47](#_Toc111240633)

[5.2. Model prediction explanation & analysis 47](#_Toc111240634)

[6. Summary 48](#_Toc111240635)

[6.1. Model performance 48](#_Toc111240636)

[6.2. Suggestions for further improvement 48](#_Toc111240637)

[7. Appendix 49](#_Toc111240638)

# 1. Overview

## 1.1. Problem

The problem associated with this problem is to generate our very own recurrent neural network capable of sentimental analysis. The problem is how the model should be trained so that it can behave much like a human mind, capable of perceiving contextual knowledge from reviews to judge whether its positive or negative in general. In which case, the project is to train a model that is able to predict accurately from 5 different categories of ratings (1, 2, 3, 4, 5). Based on various and unpredictable text humans write in their reviews, having to predict possible cases of sarcasm will be a challenge for us to overcome. At the end of the day, a model will be produced that can hopefully predict ratings with almost a ± 1 difference.

## 1.2. Objective

Build a sentiment analysis model to predict the App review score based on the Google Play Store reviews. The reviews would be based different app that we would further elucidate in the subsequent paragraphs.

## 1.3. Approach

We will be using the universal workflow of machine learning

1. Defining the problem and assemble a dataset

2. Choose a measure of success

3. Deciding on an evaluation protocol

4. Prepare our data

5. Developing a model that does better than a baseline

6. Scaling up: developing a model that overfits

7. Regularize our model and tune our hyperparameters based on performance on the validation data

1. Defining the problem and assemble a dataset

As mentioned in the problem and objective section, we have identified the uses for Sentiment Analysis and how it can help mundane tasks become more efficient in our life, and we will be assembling different dataset from the different app store reviews,

2. Choose a measure of success

A reliable measure of success to get the output of the model and determined the 4 main factors under 2 categories.

1. Training accuracy and training loss

2. Validation accuracy and validation loss

Each iteration of learning from the whole dataset will be benchmarked against its previous result giving us the value of success in the form of training success and validation success. In the first category, the training accuracy is determined after the model tried to identify all the images in the training dataset giving a percentage value of the images that it got correct while training loss is determined by the errors made by the model. The second category with validation data has the same logic as the first category with training data just that it is based on unknown data that the model has never seen before which is more applicable to the real-world context. Having a greater accuracy and a lower loss than the previous learning would be a success while the opposite will be determined as a failure.

3. Deciding on an evaluation protocol

In the given dataset, there should be sufficient data to train the model to optimal performance with a 2:8 ratio for validation and training characters but if it doesn’t, we will be using K-fold validation where it will shuffle the data and split it to k dataset and train and evaluation each set and summarises its performance, this is good for small datasets where it will train the model to identify unknown data. However, in this context, the dataset is very large and therefore does not need K-fold validation.

4. Prepare our data

We have our app review CSVs which contain raw review data, therefore before using the data in the file, we need to clean the texts to make the model learns from a good dataset so it can produce a good result. We also need to encode all the characters in the file so that the model will be able to understand it.

5. Developing a model that does better than a baseline

A baseline model for multiclass image classification will include recurrent layers and the final layer will be a fully connected layer with SoftMax activation function with an output size of each of the type of the review which is 5. For compiling the model, we will use categorical\_crossentropy. For optimizers we would be exploring more with our own models.

6. Scaling up: developing a model that overfits

A good model will be in between the range of optimization (Overfitting) and generalization (Underfitting). Using the baseline model, we can determine its performance by training over a large epoch e.g 100 epochs until it overfits and from there tune the model to prevent overfitting.

7. Regularize our model and tune our hyperparameters based on performance on the validation data

After overfitting the model, we must tune the hyperparameters or add regularizers. Here is a list of ways to fight overfitting:

1. Hold-out

2. Cross-validation

3. Data augmentation

4. L1 / L2 regularization

5. Remove layers/number of units per layer

6. Dropout

7. Early stopping

Hold-out means splitting the data into two sets: training and validation. A common split ratio is 80% for training and 20% for validation. We will train the model until it performs well on both the training and validation data. This will mean that the model can generalize well with unknown data.

Cross-validation is the same as k-fold validation but for smaller datasets and it is more computationally expensive. In our context I won’t be using k-fold as I am constrained by my computational power.

L1 / L2 regularization technique is used to constrain the model from becoming too complex by giving it a penalty in terms of the cost function to push the estimated coefficients towards zero. L2 regularization allows weights to decay towards zero but not to zero, while L1 regularization allows weights to decay to zero.

Removing layers can reduce the complexity of the model where it can better learn the data.

Dropout is a regularization technique where random neurons are ignored during training. 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.

Early stopping is when we train the model for a large number of epochs and stop the model before it starts to overfit or when it just starts too overfit. We can also get the weights of the model with the highest accuracy during training, so we won’t miss out on the best-performing model. Hence, all our members are utilizing this useful feature in the process of training.

# 2. Overview

## 2.1. Play Store app chosen, data collection & app evaluation

### 

Our team has selected the 5 apps:

* Microsoft Teams (Xihe)
* Netflix (Hasanah)
* Disney+ (Wee Kang)
* Wild Rift (Long Teck)
* Gmail (Zi Ren)

Microsoft Teams was chosen because it is an app that was made with professionals and education in mind, so reviews are more likely to be objective with minimum spam and other “dirty data” that can lower the quality of the dataset for training.

Netflix was chosen because it is a popular and widely used app around the world, thus it will have a variety of users giving the ratings which increases quantity and quality of the dataset.

Disney+ was chosen because it has a wide target audience (wide age and audience demographic), where a range of different users would use and thus possibly rate the app, increasing the variety and thus quality of the dataset.

Wild Rift was chosen because the app is catered towards a very specific target audience, where more specific data would be collected from users within this target subset, such as references and knowledge specific to a certain game.

Gmail was chosen because it is a commonly used app by semi-professionals and casual users alike as a more formal method of communication. This means that reviews would be higher in quality as users would be less likely to submit irrelevant or spam reviews.

Our team used a library called google play scraper that does most of the heavily lifting for us. Data to be collected are filtered to extract the MOST\_RELEVANT reviews as there is a lower chance of foreign languages showing up in reviews than MOST\_RECENT.

The functions of google play scraper are further described below:

The **google\_play\_scraper** will obtain data for the following columns:

* **reviewId:** Unique identifying primary key for each review
* **userName:** Username of each reviewer
* **userImage:** Profile picture of each reviewer
* **content:** Reviewer’s review content for the app
* **score:** Reviewer’s rating for the app
* **thumbsUpCount:** The number of users who tapped “Yes” on “Was this review helpful?”
* **reviewCreatedVersion:** Versioning data for the creation of the review
* **at:** dateTime data when the review was posted
* **replyContent:** Reply from the app developer to the reviewed content
* **repliedAt:** dateTime data when the replyContent to the reviewed content was posted

## 2.2. Data cleansing & data conversion to tensors

The Play Store is a public platform that is public-facing and easily accessible, where any user with a Google account would be able to post reviews for an app. This means that the reviews section is vulnerable to unworthy content and general digital waste, including but not limited to the following:

* Emojis
* Special characters not used in conventional English
* Unnecessary punctuation
* Duplicates
* Text in other languages (i.e. everything that is not found in the conventional English dictionary
* Stopwords that provide no context

These additional, nonsensical pieces of data do not help as no sentiment can be deduced from them, much less any linguistic expression. Thus, before the data is used for training, it is important to clean the data beforehand, such that the “dirty” data does not affect the training of the model to incorrectly predict reviews.

Since the problem only requires sentiment analysis of the review itself to predict the score, the only relevant columns needed are:

* **content:** Actual review data for training and testing
* **score:** To label each review with the associated score

When the 10000 reviews per rating (50000 reviews in total) have been sifted out to have the columns ‘content’ and ‘score’ only, the ‘content’ column will go through a series of text normalization techniques that will aid the model in learning more effectively.

During the cleaning, the list of reviews is iterated through. In each iteration, the words in the content are lowered as a form of normalization, misspelled words are autocorrected (not guaranteed), all use cases of punctuations and excluded stop words. Eradicating stop words is a necessary decision as usually they do not provide useful context clues for the model to learn. By filtering off extra stop words, it sets a focal point for crucial word determiners that sets the tone of the type of review.

Proceeding the above filtering steps would be a process called lemmatization. The reason to use lemmatization is a technique to normalize words. Similar to lemmatization, there is its counterpart called stemming. Stemming is a rough method whereby it chops off prefix or the suffix of a word. This results in inaccurate word forms. For example, the word ‘studies’ when stemmed will return a non-English word ‘studi’ whereby the suffix ‘es’ is removed [(Appendix – Figure 1)](#_7._Appendix_). Whereas lemmatization is an upgraded version of stemming. It converts words into their base forms that can be commonly associated during model learning [(Appendix – Figure 2)](#_7._Appendix_). This standardisation technique helps to reduce the unique word counts so the model will have deeper understanding for each word’s meanings and context.

This resource intensive process is repeated until all 50000 reviews have been looped through once. Once the data has been fully cleansed, they can finally be processed into numeric tensors for the model to understand and learn. How this process was done was through the use of tokenization. The tokenizer is initialized to take in a maximum vocabulary size of 10000 (total number of word vectors), a standard dimension size of 256 (length of word vector) and a maximum length of sequences (reviews) stored is 60 characters. The tokenizer is fitted with the entire list of dataset review contents and converted to sequences for the model to train on. Sequences lesser than 60 characters are padded accordingly so it matches the sequences matrix size.

## 2.3. Data sampling & sampling justification

The collected sequences are then shuffled to ensure that there is no bias when sampling the data. These sequences data should be sampled into three different sets of training, validation, and testing with different ratio of portions such that the size of training to validation to testing is 2:2:1. Since there are 50000 reviews in total, both training and validation datasets will have a size of 20000 while the remaining 10000 belongs to testing. Both training and validation datasets should be even as it can give us a direct comparison between training and validation performance. If the sample size of either one is lower, it would perform substantially better the other due to the probability of success increasing. Since testing is the ‘least’ resource intensive process, having a slightly smaller dataset for it will not pose much of a difference as we need a gauge on the model performance. Hence, that is the rationale for having the sample ratio 2:2:1.

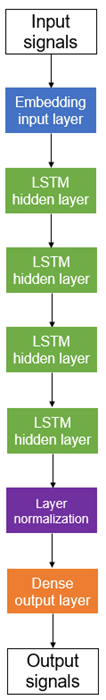
# 3. Sentiment Analysis Models

## 3.1. Han Xihe

To cut down on the amount of time spent on training, the Keras EarlyStopping callback was implemented, where a technique called early stopping is used to stop training the model before the model overfits. EarlyStopping would be used in all models unless if otherwise specified.

### 3.1.1. Model 1

The build of Model 1 is as follows:



Text

Description automatically generated

This model serves as a baseline for subsequent models to be built upon. New concepts that were introduced:

* + - **LayerNormalization():** Layer normalization is an optimization technique that is based on batch normalization. Batch normalization standardizes inputs from previous layers to the next . Layer normalization standardizes in feature direction instead, which in the context of an RNN with sentiment analysis would be more useful, as using batch normalization might negatively affect weights that are shared across the network, due to the network’s recurrent nature.
    - **unroll=True:** RNNs share the outputs from previous time steps as inputs for subsequent time steps. Unrolling modifies this nature by sharing weights as well (Brownlee, A Gentle Introduction to RNN Unrolling, 2019). This allows computation of the network to be much faster at the cost of additional memory.

The following training parameters were used:

Text

Description automatically generated

Chart, line chart

Description automatically generated

EarlyStopping was not used here and this model was trained up to 15 epochs, as the model was unable to abstract properly from the dataset with lower epochs. EarlyStopping would also not be used for model 2. From this graph, we can see that the model performs poorly, and did not manage to learn anything from the dataset with extreme underfit. However, we are looking for performance metrics as a baseline for the development of the model:

* **Training loss:** 1.6113
* **Training accuracy:** 0.1998
* **Validation loss:** 1.6098
* **Validation accuracy:** 0.2012

### 3.1.2. Model 2

This model is identical to Model 1, except that unrolling is disabled to see if unrolling provides additional gains in performance. The architecture of the model remained the same, except that unrolling was removed from the layers:

Text

Description automatically generated

Chart, line chart

Description automatically generated

Training for one more epoch before EarlyStopping stopped training, the model was still not able to learn anything from the dataset as an unrepresentative training. However, we are looking for performance metrics as a baseline for the development of the model:

* **Training loss:** 1.6112
* **Training accuracy:** 0.1940
* **Validation loss:** 1.6110
* **Validation accuracy:** 0.1964

Compared to the performance of Model 2:

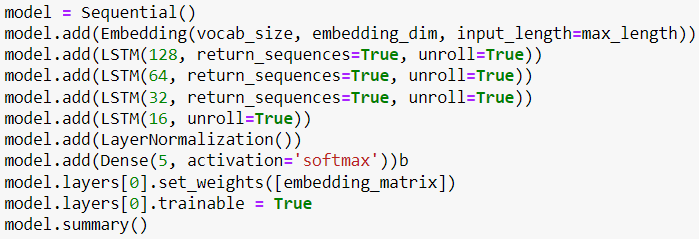
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Training Loss** | **Training Accuracy** | **Validation Loss** | **Validation Accuracy** |
| Model 1 | 1.6113 | 0.1998 | 1.6098 | 0.2012 |
| Model 2 | 1.6112 | 0.1940 | 1.6110 | 0.1964 |

Loss metrics have a very small decrease by 0.0012 for validation loss to as small as 0.0001 for training loss. However, the decrease in accuracies are much larger, with a decrease by 0.0058 for training accuracy and 0.0048 for validation accuracy. Thus, unroll would be kept true for subsequent models, as the small decrease in loss metrics is not worth the substantially larger loss in accuracies.

### 3.1.3. Model 3

To further improve the performance metrics of the model, GloVe pretrained word embeddings were added to the model. GloVe computes global word-pair co-occurrences on the dataset, creating linear representations (Pennington, Socher, & D. Manning, n.d.). While GloVe is a familiar concept, it was chosen over other word embeddings for the following benefits:

* + - **Common word weightage:** lower weights are given to words that frequently appear in the dataset. This prevents “meaningless” words from affecting the training.
    - **Word vector extraction:** while other word embeddings depend solely on the availability of local context information of words to form word vectors, GloVe utilizes word-pair co-occurrence.
    - **Word-pair context:** adding on to the way GloVe extracts word vectors, word-pair relationships are considered in the training of the model. This can help with the abstraction of more frequent phrases.

Layers of GloVe were unfrozen to allow for the use of the embedding in the network:  


Chart, line chart, scatter chart

Description automatically generated

The model is not a non-representative model, and overfits prematurely at the 3rd epoch. The following performance metrics were hit in training:

* **Training loss:** 1.0023
* **Training accuracy:** 0.5944
* **Validation loss:** 1.1827
* **Validation accuracy:** 0.5127

Compared to the performance of Model 2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Training Loss** | **Training Accuracy** | **Validation Loss** | **Validation Accuracy** |
| Model 2 | 1.6112 | 0.1940 | 1.6110 | 0.1964 |
| Model 3 | 1.6113 | 0.1998 | 1.6098 | 0.2012 |

Model 3 has a very slightly higher training loss of 0.0001, but training accuracy sees a big increase by 0.0058, validation loss decreases by 0.0012 and validation accuracy increases by 0.0048. This invalidates the minor decrease in training loss performance, thus making GloVe beneficial.

### 3.1.4. Model 4

Model 4 uses Adamax optimizer to increase the performance of the model. The architecture of the model remains unchanged, and only the optimizer is changed from RMSprop to Adamax:

Text

Description automatically generated

* + - **Adamax:** Adamax (Adaptive movement estimation max) is a variant of the Adam algorithm. Adam is an adaptive learning rate optimizer that uses two approaches of two other optimizers, AdaGrad and RMSprop. Adamax further extends on Adam by extending weight updating properties bast infinite norms of past gradients.

**Chart, line chart

Description automatically generated**

In the accuracy graph, the model peaks at the 3rd epoch before it plateaus. In the loss graph, the model overfits at the 3rd epoch with a larger generalization gap. The following performance metrics were hit in training:

* **Training loss:** 0.8850
* **Training accuracy:** 0.6356
* **Validation loss:** 1.1861
* **Validation accuracy:** 0.5369

Compared to the performance of Model 3:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Training Loss** | **Training Accuracy** | **Validation Loss** | **Validation Accuracy** |
| Model 3 | 1.6113 | 0.1998 | 1.6098 | 0.2012 |
| Model 4 | 1.0023 | 0.5944 | 1.1827 | 0.5127 |

Model 4 has substantially lower losses (by as much as 0.609 in training loss) and a very high accuracy increases, indicating that Adamax has substantial performance increases.

### 3.1.5. Model 5

A higher batch size is used here to see what happens. Batch size was increased from 32 to 256:

Text

Description automatically generated

Chart, line chart, scatter chart

Description automatically generated

The model trains for 2 more epochs and overfits at the 5th epoch, before dipping at the 8th epoch in the loss graph. In the accuracy graph, the validation line rises up to the 4th epoch until it plateaus. The following performance metrics were hit in training:

* **Training loss:** 0.9248
* **Training accuracy:** 0.6212
* **Validation loss:** 1.1198
* **Validation accuracy:** 0.5413

The higher validation loss and generalization gap of model 4 indicates that the model potentially generalizes worse. The increase in batch size from 32 to 256 aims to increase the performance when lesser noise is generated by a larger batch step. It also offers a regularizing effect in reducing the amount of overfit in the model, where the model now overfits from the 5th epoch. However, metrics see a small change.

Compared to the performance of Model 4:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Training Loss** | **Training Accuracy** | **Validation Loss** | **Validation Accuracy** |
| Model 3 | 1.6113 | 0.1998 | 1.6098 | 0.2012 |
| Model 4 | 1.0023 | 0.5944 | 1.1827 | 0.5127 |

### 3.1.6. Model 6

Model 6 reduces the learning rate of the model so that the model can learn slower to better abstract from the dataset.

A picture containing graphical user interface

Description automatically generated

The learning rate was decreased from 1e-2 to 2e-5. This reduction led to the generalization gap closing almost completely, indicating better generalization performance. Since this is the final model, it was trained with manual input for a good fit. Holding out helped the model to reach a good fit over a long number of epochs.

Compared to the performance of Model 5:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Training Loss** | **Training Accuracy** | **Validation Loss** | **Validation Accuracy** |
| Model 4 | 1.0023 | 0.5944 | 1.1827 | 0.5127 |
| Model 5 | 1.1686 | 0.5067 | 1.1930 | 0.5013 |

Since it is not worth it to train for additional epochs, it is left at 110 epochs due to diminishing returns.

## 3.2. Long Teck

My model would be trained using Word2Vec pretrained word embeddings. This model has a vector size of 100 and a vocabulary size of 4,027,169. The Word2Vec word embeddings are from the English CoNLL17 corpus. The algorithm used for this Word2Vec word embedding is continuous skipgram.

Diagram

Description automatically generated

Figure 3.2.1 – Skipgram architecture

Figure 3.2.1 shows the Skipgram architecture. Skipgram tries to predict several context words from a single input. w(t) is the word input. The hidden layer performs dot product between weight matrix and the word input. As such the output layer is the value of the dot product. Lastly, the architecture would then apply softmax to compute probability of words appearing in the given context. Since Skipgram rely on single word inputs, it is less sensitive to overfit frequent words.

For the models that I am going to be training, the loss function would be categorical\_crossentropy as the problem type is multiclass, single-label classification. The models would also have a final Dense layer with 5 units as there are 5 reviews values. The activation function for the final layer would be softmax.

Moving on to building my first model, I would adding random units to my Bidirectional GRU layers to try and get my model to overfit following the machine workflow. Hence, my first model is made up of 4 Bidirectional GRU layers with units of 512, 256, 128, 32. The model would be trained with RMSprop optimizer with a learning rate of 2e-5. The model would be fitted for 100 epochs with a batch size of 128. I have also implemented keras callback earlystopping feature with a patience of 10 and it would monitor the validation accuracy. As such, if the validation accuracy decreases for more than 10 epochs, the model would stop training.

### 3.2.1. Model 1

Graphical user interface, chart

Description automatically generated

Figure 3.2.1.1 - First Model Results

As seen in Figure 3.2.1.1, the model attained a final accuracy of 45.46%. The model also seems to have overfitted at around the 70th epoch. With the addition of earlystopping I would not have to train my models for as long, thus allowing me to experiment more with the different hyperparamters for my model.

For my next model, I would also be decreasing the learning rate to 1e-5 as I would like to see if my model would perform better with a lower learning rate. Having a lower learning rate would also help reduce the n=noise as seen in my current validation accuracy and loss.

### 3.2.2. Model 2

Graphical user interface, chart, histogram

Description automatically generated

Figure 3.2.2.1 - Second Model Results

From Figure 3.2.2.1, the model attained a final accuracy of 45.22% and it seems that the model overfitted at around the 90th epoch. With the decrease in learning rate, the model’s validation loss and accuracy are also less noisy.

However, I would be using the value 2e-5 for the learning rate as it seems that my model had achieved a higher accuracy. So, for my next model, I would be switching the Bidirectional GRU layers with Bidirectional LSTM layers. My intention is to compare which layer would give a higher accuracy.

### 3.2.3. Model 3

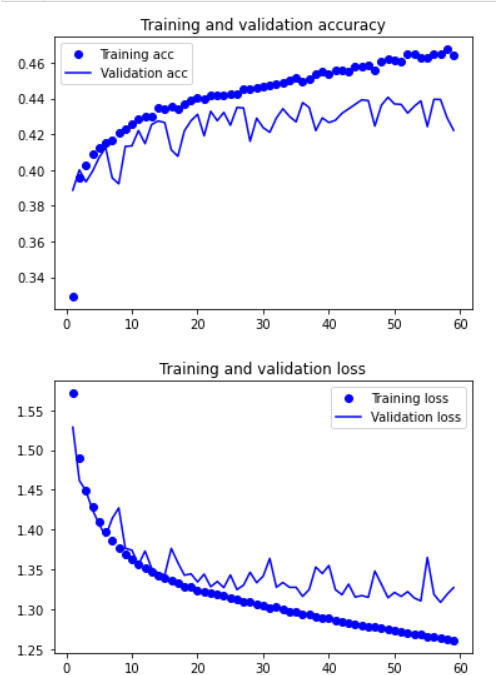


Figure 3.2.3.1 - Third Model Results

As seen in Figure 3.2.3.1, the model stopped at the 59th epoch due to early stopping and the model attained a final accuracy of 42.22%. However, looking at the validation loss, the model does not seem to be overfitting. On the other hand, if I look at the validation accuracy, the model did not achieve the same or better accuracy of 44.06%, as such early stopping is called, and my model stopped training at the 59th epoch.

After training with Bidirectional LSTM layers, I would be training with Dense layers next. As seen in the weekly practical, Dense layer were used to do sentiment analysis as well. As such, I would be testing out in this assignment whether Dense layer is effective or not.

### 3.2.4. Model 4

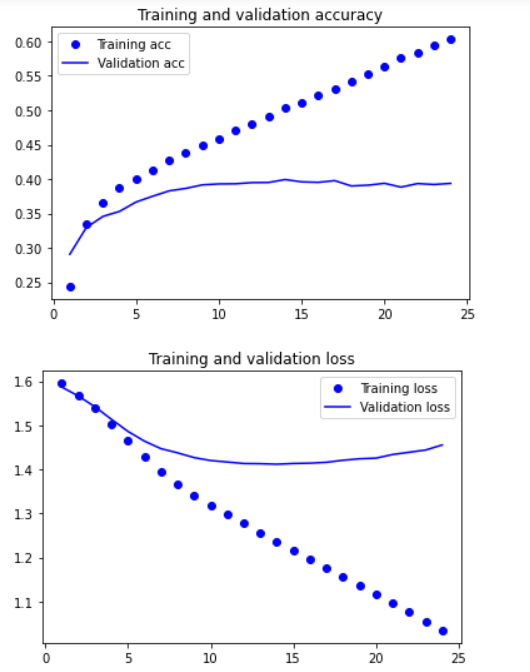


Figure 3.2.4.1 - Fourth Model Results

As seen in Figure 3.2.4.1, the model attained a final accuracy of 39.37% and model overfitted at around the 15th epoch.

Comparing the 3 models with different layers, Bidirectional GRU, Bidirectional LSTM and Dense layer, it seems that the model with Bidirectional GRU layers performed better with an accuracy of 45.22%. As a result, for my following models, I would be training it with Bidirectional GRU layers in hope that the accuracy of the model would be the same or even better. Additionally, the models would be trained with a learning rate of 2e-5, as from the previous comparison, it had a higher accuracy.

For my next model, I would be experimenting with the optimizer. Specifically, I would be trying out the Adam optimizer. Using the first model as a base, suggesting that for my fifth model, the layers and hyperparameters would be kept the same as the first model except the optimizer.

### 3.2.5 Model 5

Graphical user interface, chart, histogram

Description automatically generated

Figure 3.2.5.1 - Fifth Model Results

From Figure 3.2.5.1, you can see that the model had achieved a final accuracy of 46.07%. It seems that the model is beginning to overfit at around the 80th epoch, however, it is not so obvious as if I trained the model for more epochs, the validation loss might continue to decrease, which would suggest that the model is underfitted. Similarly, looking at the overall trend of the validation accuracy, it is still increasing. As such, I would assume that the model currently is still underfitted.

To counter the underfitting issue, for my next model, I would be adding more layers and changing the units of some of the pre-existing layers. Originally, I had 4 Bidirectional GRU layers with units of 512, 256, 128 and 32. After adding and changing the layers and units, I have a total of 5 Bidirectional layers with units of 512, 512, 512, 256, 32.

### 3.2.6 Model 6

Graphical user interface, chart, line chart

Description automatically generated

Figure 3.2.6.1 - Sixth Model Results

As seen in Figure 3.2.6.1, it seems that my model overfitted at around the 50th epoch. My model achieved a final validation accuracy of 45.59%. I am please with the result as the overfitting is more obvious as compared to the previous model as seen in Figure 3.2.5.1.

Since my model have overfitted, now, I would need to add either regularizers or dropout layers to solve the overfitting issue. For my model I added SpatialDropout1d to my model. Specifically, I added 2 SpatialDropout1D layer both with values of 0.4. SpatialDropout1D performs the same functions as Dropout, however, it drops the entire 1D feature maps instead of individual elements. SpatialDropout1D helps promote independence between feature maps as the adjacent frames within the feature maps. SpatialDropout1D performs variational dropout within the model, which should help my model generalise better by having the model overfit at a later epoch.

### 3.2.7 Model 7

Graphical user interface, chart

Description automatically generated

Figure 3.2.7.1 - Seventh Model Results

From Figure 3.2.7.1, the model overfits later at around the 80th epoch. My model also attained a final validation accuracy of 45.59%. As such, the optimal range, whereby the model starts to overfit, is at around epoch 80.

As such, for my next model, keeping all the layers and hyperparameters the same as the previous model, I would be training it until the 80th epoch.

### 3.2.8 Model 8

Graphical user interface, chart

Description automatically generated

Figure 3.2.8.1 - Eighth Model Results

Figure 3.2.8.1 displays my final model. The model attained a final accuracy of 45.67%.



Figure 3.2.8.2 – reviews with slang(fps)



Figure 3.2.8.3 – reviews with slang(tfc)

Text

Description automatically generated

Figure 3.2.8.4 – Foregin reviews

One reasons that might explain why my model attained such poor accuracy is because the data might still be a bit “dirty”. As my reviews are based off a game, there might be a lot of game slangs used in the reviews. As seen in Figure 3.2.8.3 and 3.2.8.4, the reviews shown have word slangs or abbreviations in it. As these slangs might not be in the word embeddings it might throw off the model during the training process.

Another example of “dirty” data are reviews that written with foreign words. As seen in Figure 3.2.8.4, the reviews are most likely in Malay. The data processing only filters out reviews that are not written with English words/letters. However, because Malay language uses English letters as well, our data processing is unable to filter out these reviews. Since the pretrained word embedding only has English words, there are no pretrained word embeddings for these malay words.

As a result, due to these dirty data, the model is not able to generalise properly, resulting in such poor accuracy.

As such, an improvement that can be made to my model is to clean he data even more thoroughly, such that only English words in the dictionary are present in the reviews. Another improvement that can be made is to experiment more with the different type of layers and hyperparameters and find the configurations to attain a better accuracy.

## 3.3. Hasanah

### 3.3.1. Model 1

Following the universal workflow machine learning, a baseline model will first be developed. After defining the model using sequential, an Embedding layer takes in vocab\_size of 10 000 (number of words), embedding\_dim of 128 (the dimensionality of the embeddings) and input\_length of 60 (length of sequence) as its parameters. This is followed by another two GRU layers with 32 filters each. The first GRU layer will set return\_sequences=True so that the second GRU layer has a three-dimensional sequence input. The last layer will be a dense layer of size 10 and a softmax activation. Softmax activation is the most suitable as the problem type is a categorical classification. As a result, the summary of the model will show that there are a total of 1 302 053 parameters.

Text

Description automatically generated

Figure 3.3.1 - Build Model

From the figure below, the model will then be compiled using the loss of categorical crossentropy, Adam optimizer with a default learning rate set to 1e-4 and accuracy as its metrics. Next, the model will be trained by calling the model.fit() function. It will be trained using the batch size and the number of epochs set to 20. Next, the loss and accuracy of model over training and validation data will be plotted.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 3.3.2 - Compile and Fit Model

The figure below shows the result of plotting the training and validation accuracy and loss. There is early overfitting where the validation loss starts to increase starting from the 3rd epoch. The model has a very low validation accuracy of 0.44 maximum and a high validation loss of 1.3 minimum. As it starts to overfit, there is a large margin between the training and validation graphs.

Chart

Description automatically generatedChart, line chart

Description automatically generated

Figure 3.3.3 - Plot results

### 3.3.2. Model 2

As the model is already overfitting, there is no need to scale up the model until it overfits. Thus, the second model is developed to reduce overfitting and regularise the model. One of the techniques to reduce overfitting is by adding in dropout regularisation. As shown from the figure below, two types of dropout regularisation are added in each of the GRU layer. They consist of a normal dropout which randomly drops the inputs of the layer as well as a recurrent\_dropout which is a special dropout for recurrent layer only where the dropout happens within recurrent unit between different Sates.

Text

Description automatically generated

Figure 3.3.4 - Add Dropouts (Second model)

As a result, the model is still overfitting starting from the 5th epoch. The accuracy and loss of the model remains the same at 0.44 maximum and 1.3 minimum respectively.

Chart

Description automatically generatedChart

Description automatically generated

Figure 3.3.5 - Plot Results (Second Model)

### 3.3.3. Model 3

To reduce overfitting further, the model will be regularised by tuning its hyperparameters. In the third model, the hyperparameter to be tuned is the learning rate which will be changed from 1e-4 to 1e-5.

Text

Description automatically generated

Figure 3.3.6 - Tune Learning Rate (Third Model)

As shown in the figure below, the model is no longer overfitting. The validation and training graphs are also very similar as the margin has been narrowed. However, the performance of third model compared to the baseline model has slightly deteriorated as the accuracy has decreased from 0.45 to 0.4.

Chart

Description automatically generatedChart, line chart

Description automatically generated

Figure 3.3.7 - Plot results (Third model)

### 3.3.4. Model 4

When developing the fourth model, the learning rate will be set back to its default value which is 1e-4 so as that its performance will not worsen. This time, the hyperparameter that will be tuned is the optimiser. Instead of using Adam, RMSprop will be chosen as the optimiser.

Text

Description automatically generated

Figure 3.3.8 - Change Optimiser (Fourth Model)

Consequently, after plotting the graphs, there are no problems of overfitting or underfitting. Compared to the third model, the validation accuracy has slightly increased from 0.45 to 0.46 whereas the validation loss has decreased from 0.3 to 0.25. Therefore, the model’s performance has improved.

Chart

Description automatically generatedChart

Description automatically generated

Figure 3.3.9 - Plot Results (Fourth Model)

|  |  |  |
| --- | --- | --- |
| **Models** | **Validation Accuracy(%)** | **Validation Loss** |
| 1 | 44.54 | 1.2956 |
| 2 | 45.01 | 1.2898 |
| 3 | 39.20 | 1.3475 |
| 4 | 45.62 | 1.2580 |

Figure 3.3.10 – Performance Comparison

As shown in the figure 3.3.10, the fourth model has the highest accuracy of 45.62 and the lowest validation loss of 1.2580. It uses 2 GRU layers, 4 dropouts in total, RMSprop optimizer with learning rate set to 1e-4 and 20 as the batch size and number of epochs. Therefore, it is chosen as the best model as it has the best performance out of all the models.

## 3.4. Wee Kang

The models will be trained using a reputable pre-trained embedding from 2-million-word vectors consisting of 600 billion tokens trained on Common Crawl, an open repository of web crawl data . The vectors have the dimensions of 300 (length of the vector), which is on the larger side, giving more substance for the model to train on. This word vectors have the architecture of fastText, which is an efficient, open-sourced library for learning of word embeddings and text representations produced by the reputable Facebook’s AI and research lab. fastText supposedly has a better performance than the other counterpart Word2Vec, though it takes a longer time to train a fastText model. The word embeddings will be instantiated and set as the model’s primary weights.

Text

Description automatically generated

**Figure 3.4.1 – Python code to load word vectors**

At the start of all models, the embedding matrix will take on a vocab size of 10000 words, a dimension of 300 since it has to be similar to the dimensions of the fastText pretrained word vector. These configurations make sure that the loaded word embeddings are appropriately fitted to the correct settings of the pretrained word vectors.

Models will be using the learning optimizer Adam. Adam is a popular type of optimizer for stochastic gradient descent. It is the best among the adaptive optimizers in utilizing the concept of momentum. Some of the benefits to using Adam optimizer are:

1. Easy to implement
2. Computational efficient
3. Reduce noisy gradients (smoother learning)
4. Requires little hyperparameter tuning

A batch size of 32 is used as it achieves a balance between generalization and computational resources. Though it may be favourable to increase the batch size for a quicker training process as it converges faster, there will be a significant degradation in the performance of the trained model in terms of its generalization capability. The batch size 32 is a recommended starting point, and it has showed promising results which are to be expounded on in the trained model sections later.

### 3.4.1. Model 1

The first model will first use two LSTM layers of 64 and 128 nodes. It is compiled with a learning optimizer Adam of 1e-4 with a batch size of 32. The implementation of the model is shown in **Figure 3.4.1.1** below.

Graphical user interface, text

Description automatically generated

**Figure 3.4.1.1 – Model 1 layers**

Chart, line chart, scatter chart

Description automatically generatedChart, line chart

Description automatically generated

**Figure 3.4.1.2 – Model 1 performance**

The model overfits at a quick epoch 14, reaching a validation accuracy of 51.82% and a validation loss of 1.1264 shown in (**Figure 3.4.1.2**). Both training and loss curves are not as stable, fluctuating every now and then. This shows the baseline performance of the LSTM layer.

### 3.4.2. Model 2

In my model 2, LSTM layers are replaced with the usage of GRU to compare which layer is more fitting in this application of predicting app reviews. The number of layer and nodes are the same, compiled with the same batch size of 32, same optimizer and learning rate of 1e-4. This implementation is shown in **Figure 3.4.3.1** below**.**

Graphical user interface, text

Description automatically generated with medium confidence

**Figure 3.4.2.1 – Model 2 layers**

Chart, line chart, scatter chart

Description automatically generatedChart, line chart

Description automatically generated

**Figure 3.4.2.2 – Model 2 performance**

The model 2 performs slightly better by 0.18% validation accuracy and a lower validation loss of 0.0093 (**Figure 3.4.3.2**). It also overfits later by 3 epochs than model 1 (**Figure 3.4.3.3**).. These are the results of a fundamental GRU model.

Comparing between This means that GRU has a better performance than LSTM when capable of generalizing well with the diverse and unpredictable human reviews.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model​** | **Layers​** | **Validation accuracy (%)​** | **Validation loss​** | **Overfit (Epoch)​** |
| **1​** | 2 LSTM (64, 128)​ | 51.82​ | 1.1264​ | 14​ |
| **2​** | 2 GRU (64, 128)​ | 52.0​ | 1.1171​ | 17​ |

**Figure 3.4.2.3 – Comparison between model 1 and 2**

### 3.4.3. Model 3

This model will feature the use of bidirectional GRU. In short, bidirectional layers help to transfer learnable information in two directions between hidden layers. This helps the model to learn past and future information which would increase it’s understanding on context the dataset. The number of layers and nodes will still be similar to that of model 2, with the additional wrapping of bidirectional layers around the GRU layers. Since it was observed that model 2 overfits at an early 17 epochs, adding more parameters will surely overfit faster. Hence, the use of regularization layer called SpatialDropout1D of 0.2 is used. The SpatialDropout1D is placed after every GRU layer to drop random channels after learning shown in **Figure 3.4.3.1**.

Table

Description automatically generated

**Figure 3.4.3.1 – Model 3 layers**

Chart

Description automatically generated Chart

Description automatically generated

**Figure 3.4.3.2 – Model 3 performance**

Due to the very low learning rate, the model is underfitting despite having a more complex structure to it (Bidirectional layers). Nevertheless, it makes sense that a slower learning rate and addition of SpatialDropout1D. Its highest validation accuracy reached is 53.57 and lowest validation loss of 1.09 (**Figure 3.4.3.2**). More regularization layers need to be implemented in the subsequent models.

### 3.4.4. Model 4

Compared to model 3, a different regularization layer called Layer Normalization is used instead of SpatialDropout1D to see if their effectiveness in regularizing while maintaining a good performance. Model 4’s layers will be the same as model 3 just that the regularization layers are alternating.

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

**Figure 3.4.4.1 – Model 4 performance**

Model 4 also underfitted just like model 3 due to the slow learning rate of 5e-5 and its regularization layers of Layer Normalization. Comparing between model 3 and 4 (**Figure 3.4.4.2**), despite both models being underfitted, model 4 has a lower validation accuracy of 51.81 and higher validation loss of 1.1265 (**Figure 3.4.4.1**). So for the choice of regularization layer, SpatialDropout1D will be used in future models.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model​** | **Layers**(Embedding layer using fastText word vectors)**​** | **Validation accuracy (%)​** | **Validation loss​** | **Overfit (Epoch)​** |
| **3​** | 2 Bidirectional GRU(32, 64), 2 **SpatialDropout1D**(0.2),​  GlobalMaxPooling1D​ | ​  53.57​ | ​  1.0900​  ​ | -​ |
| **4​** | 2 Bidirectional GRU(32, 64), 2 **LayerNormalization**,​  GlobalMaxPooling1D​ | 51.81​ | 1.1265​ | -​ |

**Figure 3.4.4.2 – Comparison between model 3 and 4**

### 3.4.5. Model 5

Model 5 has the same architecture of model 3. However, the weights are allowed to train so it can better generalize on my dataset. It is trained with the same learning rate of 5e-5 just like model 3.

Chart, scatter chart

Description automatically generated Chart

Description automatically generated

**Figure 3.4.5.1 – Model 5 performance**

Comparing between model 3 and model 5 that both uses the same layers (**Figure** **3.4.5.2**), model 5 overfits much faster at epoch 13 despite the low learning rate. Despite model 3 underfitting, it still has a higher validation accuracy by 0.33% and lower validation loss by 0.0134 (**Figure 3.4.5.1**) than model 5. So, future models will have their weights fixed so that the rich calculations from the pretrained model are preserved.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model​** | **Layers​** | **Validation accuracy (%)​** | **Validation loss​** | **Overfit​**  **(Epoch)​** |
| **3​** | 2 Bidirectional GRU(32, 64), 2 SpatialDropout1D (0.2),​  GlobalMaxPooling1D​ | ​  53.57​ | ​  1.0900​  ​ | -​ |
| **5​** | 2 Bidirectional GRU (32, 64), 2 SpatialDropout1D (0.2),​  GlobalMaxPooling1D **(Trainable weights)**​ | 53.24​ | 1.1034​ | 13 |

**Figure 3.4.5.2 – Comparison between model 3 and 5**

### 3.4.6. Model 6

Since it was observed that the learning rate in model 3 was too slow (5e-5), the learning rate was raised back to 1e-4 in model 6, whilst adding a few more nodes into the same number of GRU layers to ensure that it overfits at an acceptable point.

Chart, line chart

Description automatically generated Chart, line chart

Description automatically generated

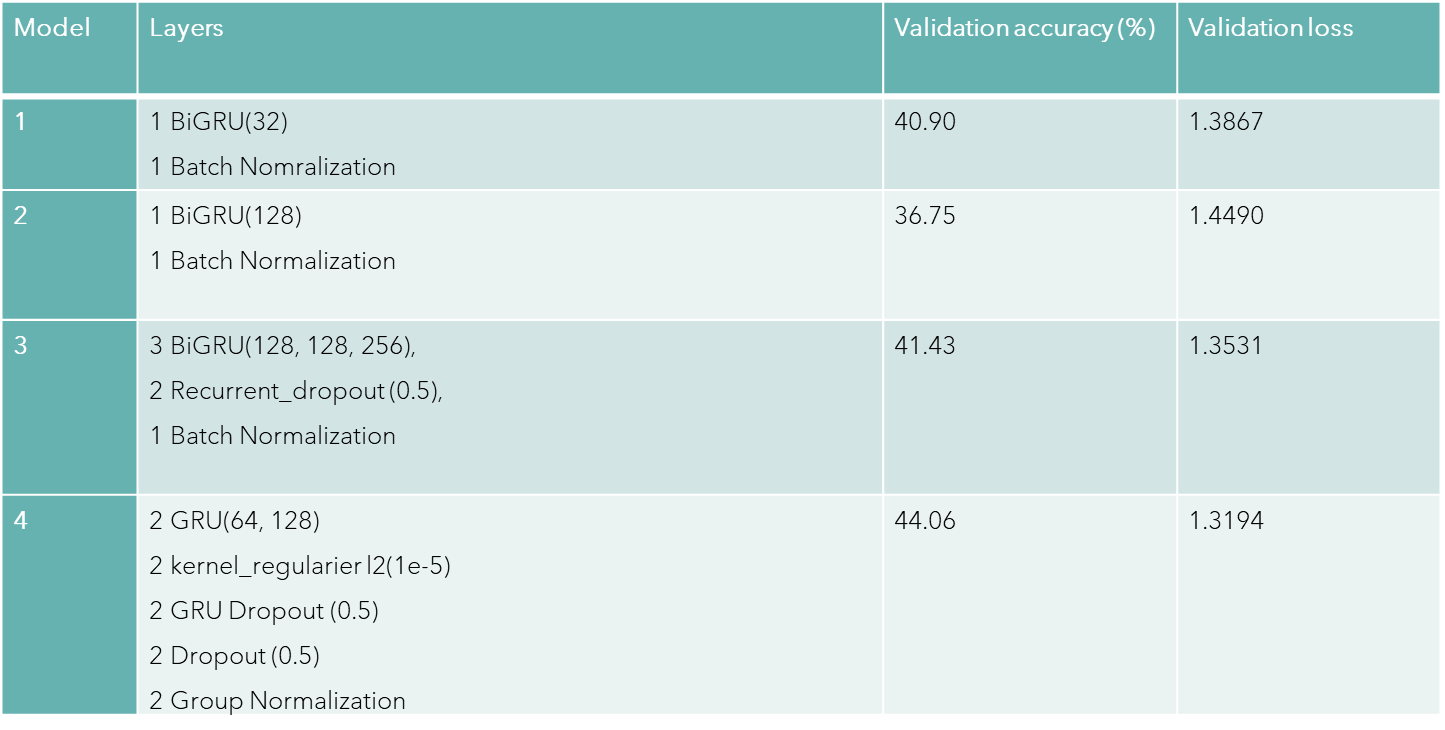
**Figure 3.4.6.1 – Model 6 performance**

Model 6 overfits at epoch 17 as opposed to model 3 underfitting state (**Figure 3.4.6.2)**. This is to be expected since there are greater number of parameters, thus the model learns more effectively and quickly. Model 6 reaches the highest validation accuracy of 53.65% compared to all other 5 models and the lowest loss of 1.0817 shown in **Figure 3.4.6.1**. This will be the best performing model trained using pretrained fastText Common Crawl word vectors. Much more could be improved such as the usage of stronger pretrained word vectors.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model​** | **Layers​** | **Validation accuracy (%)​** | **Validation loss​** | **Overfit​ (Epoch)** |
| **3​** | 2 Bidirectional GRU(**32, 64**), 2 SpatialDropout1D (0.2),​  GlobalMaxPooling1D​  ​  **Learn rate = 5e-5**​ | ​  53.57​ | ​  1.0900​  ​ | -​ |
| **6​** | 2 Bidirectional GRU(**64, 128**), 2 SpatialDropout1D (0.2),​  GlobalMaxPooling1D​  ​  **Learn rate = 1e-4**​ | 53.65​ | 1.0817​ | 17​ |

**Figure 3.4.6.2 – Comparison between model 3 and 6**

## 3.5. Yong Zi Ren



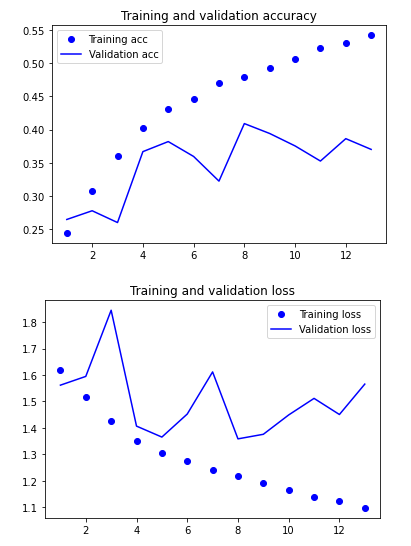
This is the overview of the progression of the model that I have built. It started off at 40.9% accuracy all the way to 44.06% in the final model.

Text

Description automatically generated

This is the code to train the model, it has a batch size of 32, for the model to learn more specific feature, the callback will be earlystopping with 10 epochs as patience and it will save the best weights in training, the metric to monitor will be the validation accuracy as that is the actual accuracy when the model is learning unknown data. This means that the model will stop learning after there is no increase in the validation accuracy in 10 epochs after the highest accuracy and revert the model to the state with the highest validation accuracy, reason for choosing 10 compared to a lower value or higher value is that if a lower value is chosen, there is a possibility that the model has not fully converged and stopping it would make the model not able to achieve its best potential while choosing a higher epoch will just be wasting time as after 10 epochs the model most likely have overfitted.

### 3.5.1. Model 1



Graphical user interface, text, application

Description automatically generated

Model 1 was my baseline model after doing some research on building RNN models for sentimental analysis, I started off with a bidirectional GRU layer as GRU is much less computationally expensive than LSTM while also giving similar results with a 32-memory unit size similar to my batch size and lastly a batch normalization layer to deal with noisy learning curve as most of the reviews can be drastically different from one another. The model hit the highest validation accuracy of 40.9% and has the lowest validation loss of 1.3867. Overall, the model is still very noisy and starts to overfit around the 8th epoch.

### 3.5.2. Model 2

Chart, line chart

Description automatically generated

Text

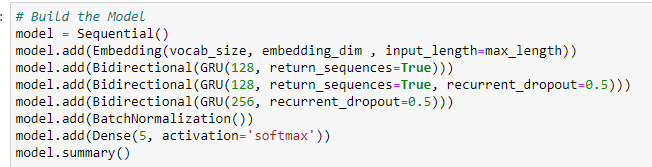
Description automatically generated

For model 2 I decided to increase my memory unit size from 32 to 128 to see if learning with more data at once can give a higher accuracy or not. From the table we can see that the highest validation accuracy decreased to 36.75% and the validation loss also increased to 1.449. Overall, the model become even noisier than model 1 and achieved a worse result, overfitting at the 5th epoch.

### 3.5.3. Model 3

Chart, line chart, scatter chart

Description automatically generated



For model 3 I decided to add 2 more bidirectional gru layers with 128 and 256 memory unit size while also adding a recurrent dropout of 0.5. This change was to hopefully reduce overfitting while also generalized the data better. From the table we can see that the validation accuracy did indeed rise to 41.43% compared to 40.9% in model 1 and the validation loss also decreased from 1.3867 in model 1 to 1.3531. Overall, we can see that the model became less noisy and starts to overfit at a later epoch at about the 8th epoch.

### 3.5.4. Model 4

Chart, line chart, scatter chart

Description automatically generated

Graphical user interface, text, application

Description automatically generated

For model 4, I removed the bidirectional as it was not really helping the model that much and introduced more regualrizers like kernel regulaizers and gru layer dropouts and 2 dropout layers to help fight the overfitting. I also introduced group normalization layer which is similar to layer normalization just that it further divides the layers into groups and normalized it. Overall, the model became less noisy and achieved the highest validation accuracy of 44.06% while having the lowest validation loss of 1.3194.

# 4. Model Evaluation using Testing Data

## 4.1. Model performance analysis during testing phase

During the testing phase, other team members’ dataset has been used to test on each of the members’ best model. The expectation was for our model to perform worse as compared to testing it against our own dataset.

## 4.2. Model performance comparison & evaluation

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of member’s model** | **Name of member’s dataset** | **Validation accuracy (%)** | **Validation loss** |
| Wee Kang | Hasanah (Netflix) | 20.06 | 2.0610 |
| Xihe | Zi Ren (Gmail) | 19.44 | 1.9380 |
| Long Teck | Wee Kang (Disney+) | 23.32 | 1.9110 |
| Zi Ren | Long Teck (Wild Rift) | 20.54 | 2.4709 |
| Hasanah | Xihe (MSTeams) | 21.06 | 2.0146 |
| Average accuracy: |  | 20.88 | 2.0791 |

Figure 4.2 – Model Performance Comparison

As shown from the figure above, all the validation accuracy are below thirty percent whereas the validation loss values fall in between the range of 1.9 and 2.5 when using another member’s dataset. The average validation loss calculated from the mean of all the validation loss values is 2.0791 which is a very high value signifying that there are many errors generated in the model when tested on another dataset. The validation accuracy is very low thus it is difficult for the models to make a good and accurate prediction of the App Review score. We were shocked to see our model perform so poorly when tested on a different dataset.

A possible reason for the bad performance of all the models is that each model has been trained and tested multiple times on its own dataset, thus giving a good prediction and result. However, when the model sees and is fed a different and unfamiliar dataset, it is not able to predict accurately. This suggests that the models are not generalised enough as once it was given a different review, it could not predict it correctly. Within different dataset, although highly likely as our data is made up of 10,000 reviews per rating, there might be words used that the model might not have learnt about. The word used might be so influential that it changes the entire context of the review to a positive one although the other words give the influence that the review is a negative one. As a result, all our model performed worse when using another dataset.

## 4.3. Best model & why it is the best

The best model chosen out of the five models is Long Teck’s model which is tested on Wee Kang’s dataset. It has the best performance as it has the highest validation accuracy of 23.32% and the lowest validation loss of 1.9110. The model might have the best accuracy when tested on a different dataset might be because of the similarity of the type of reviews given. The reviews might have been similar which resulted in the model predicting more reviews accurately than the rest of the models. Since the model had the best accuracy when tested on a different dataset, we came into a consensus and agreed to use the model to be tested on real life inputs.

# 5. Prediction with the Best Model

## 5.1. How the model can be applied to real-life text inputs

Using the best model for sentimental analysis helps to predict any type of review be it apps, movies, or food reviews. The model can convert any text feedback into a calculatable metric to able to draw more insights into how feedback can be algorithmized to be used as a quantitative success metric.

## 5.2. Model prediction explanation & analysis



When we first tested a negative review with the model, we were quite surprised by the fact that it was almost accurately able to correctly identify it as a bad review with a 2-star rating, keeping in mind that the model had 23% accuracy. In a world with the optimal model, this review would be a 1-star review but getting a 2-star is already very close to the target.

However, when tested on positive reviews, it also gave the same 2 stars review which should the actual result with a model that has 23% accuracy.

A possible reason for this is that some reviews contradict themselves or the user is just playing around, purposely giving the app a bad review even though the review given is a fantastic one. Given an optimal model, we would want the review to be 5 stars and since the model labelled it as 2-stars we are still far from the optimal model.

# 6. Summary

## 6.1. Model performance

In summary, the best model showed a promising validation accuracy of 45.67%. The best model performed a shockingly low testing accuracy of 23.32% despite the use of an efficient word2vec with the architecture of skip gram.

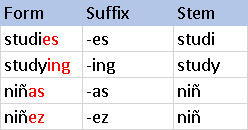
## 6.2. Suggestions for further improvement

In the future, we might want to try out other different pre-trained word vector models which may give a higher accuracy as the pre-trained model saves a lot of time to achieve results that might take hours of training if started from scratch.

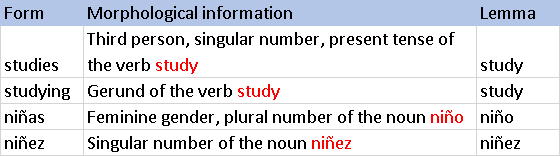
We might want to consider other layers to implement like Group Normalization, MaxPooliong1D or Attention layer which help to look at all hidden states from encoder sequence for making predictions unlike vanilla Encoder-Decoder approach. In a simple Encoder-Decoder architecture the decoder is supposed to start making predictions by looking only at the final output of the encoder step which has condensed information. On the other hand, attention-based architecture attends every hidden state from each encoder node at every time step and then makes predictions after deciding which one is more informative. We can also try to fit the model with other optimizers like AdaBelief or AdamW.

If we have unlimited computational power, we can try out keras\_tuner which uses keras tuner Bayesian optimizer to automatically tune the hyper parameter, so it takes lesser time to fine-tune and trial and error manually to find the perfect learning rate or the memory unit size for each of the layers.

# 7. Appendix



**Figure 1 –** [**Stemming example**](https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/)



**Figure 2 –** [**Lemmatization example**](https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/)