Assignment 3
Sentiment Analysis with basic neural networks

Name: Gopi Trinadh Maddikunta **PSID:** *2409404*

Sentiment Analysis is a technique which is used to describe the tone of an emotion or in another meaning extract the subject meaning of text under three different categories such as positive, neutral, negative. By interpreting these sentences will help us understand the behavior of that specific domain.

The main objective of this assignment is to develop neural network models to classify the sentiment of short financial sentences into three categories. They are

negative (0)
neural (1)
positive (2)

The dataset consists of 5,482 [but I got 5,842] short financial sentences with sentiment category as stated above. This dataset supports the goal which helps to classify the each sentence and is used to train various neural network models using pretrained Word2Vec embeddings.

Let us proceed with step 1: Preprocessing, each sentence in dataset is cleaned and standardized. So, that helps to ensure compatibility with pretrained Word2vec embeddings. Cleaning processing starts with all numerical sequences including decimal parts are replaced with single digits [1, 2, 3, 4, 5, 6, 7, 8, 9] to match Word2Vec's vectors. For instance, "15%" to "3%". And then replacing the financial terms such as 'k', 'm', 'b', and 't' following numeric values are expanded to 'thousand', 'million', 'billion' and 'trillion' respectively, while monetary units like 'USD7m' or 'CAD 2T' are modified to place the currency at the end as '7 million USD' or '2 trillion CAD' respectively, which can handles the both types, spaced and unspaced. At last I have removed all punctuations and special characters and follows some conditions like no lemmatization, stemming, stop words removal, helps in preserving the linguistic richness [*this word found in google*] which adds weight to sentiment analysis.

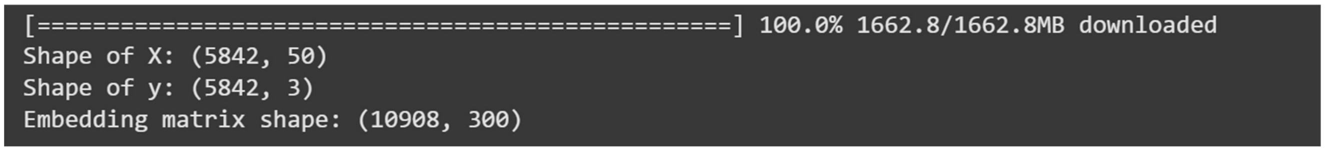
Result:(from step 1)

$ESI on lows, down $1.50 to $2.50 BK a real possibility

to

ESI on lows down 7 to 8 BK a real possibility

In step 2: Creating an Embedding matrix, I have create a reduced embedding matrix according to that dataset by using the pretrained Word2Vec Google News 300 model [word2vec-google-news-300] via the genism.downloader interface, during this process I have faced lot of issues with modules, to use these I needed to downgrade the versions, it may my system faults too. Next, I have constructed a vocabulary using Keras's Tokenizer, ensuring that only tokens present in the Word2Vec vocabulary are retained. The first token in this vocabulary is manually set to </s> at index 0, serving as sentence separator. For each token in vocabulary, it corresponds to 300-dimensional Word2Vec vector is extracted and stored in a new embedding matrix, it helps to reduce the memory drastically instead of loading all 3 million vectors. Then each input sentence is transformed into a sequence of indices corresponding to this vocabulary which is zero padded so, the length of sequence matches the maximum sentence length + 2 tokens. Eventually, Sentiment labels (0, 1, 2) are converted into one-hot encoded vectors using Keras's to_categorical function, helps in producing the target matrix.



```
[==================================================] 100.0% 1662.8/1662.8MB downloaded
Shape of X: (5842, 50)
Shape of y: (5842, 3)
Embedding matrix shape: (10908, 300)
```

Fig: Describing the shapes

The above figure shows that I have created the total number of tokens is 10,908 including **</s>** where each token corresponds to pretrained model and retrieved the embedding matrix shape (10908, 300) and shape of input matrix **X** (5842, 50) where 50 is the longest padded sentence and giving label matrix **y** with the shape of (5842, 3).

Let us dive into Step 3: Neural networks, firstly, I have evaluated whether the dataset suffers from class imbalance, a histogram of sentiment labels is plotted. The calculated imbalance ratio is 3.64, indicating that the neutral class dominates the dataset significantly. To address this, class weights were computed and applied during training to mitigate bias towards the majority class. The weights used were: [0: 2.26, 1: 0.62, 2: 1.05], ensuring balanced learning across all sentiment categories.

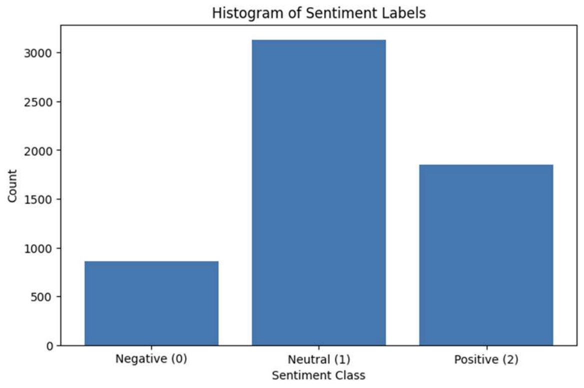
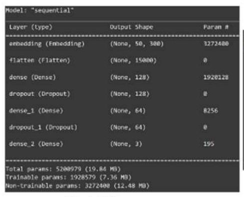


Fig: Histogram

1. FFN Classifier with pre-trained embeddings:

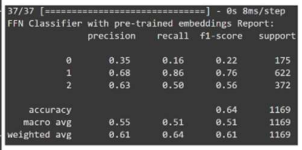
This model with pre-trained Word2Vec embeddings (frozen) was implemented using keras with two hidden dense layers (128 and 164 units) and dropout for regularization. This model was trained using categorical cross-entropy loss and early stopping. Result training accuracy (~90%), validation accuracy (~62%) and lose indicates overfitting. Final test accuracy (64%) was performed well on neutral class, negative class effect with class imbalance. The model has 5.2m parameter, 3.27m are non-trainable means embeddings are in frozen mode.

FFN Classifier with pre-trained embeddings



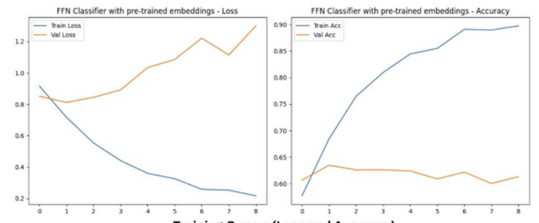
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 300)           3272400

 flatten (Flatten)           (None, 15000)             0

 dense (Dense)               (None, 128)               1920128

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 3)                 195

=================================================================
Total params: 5200979 (19.84 MB)
Trainable params: 1928579 (7.36 MB)
Non-trainable params: 3272400 (12.48 MB)
```

Model Summary



```
37/37 [==============================] - 0s 8ms/step
FFN Classifier with pre-trained embeddings Report:
              precision    recall  f1-score   support

           0       0.35      0.16      0.22       175
           1       0.68      0.86      0.76       622
           2       0.63      0.50      0.56       372

    accuracy                           0.64      1169
   macro avg       0.55      0.51      0.51      1169
weighted avg       0.61      0.64      0.61      1169
```

Classification Report

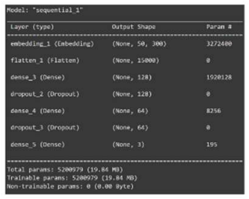


Training Curves (Lose and Accuracy)

2. FFN Classifier with Fine-tuning pre-trained embeddings:

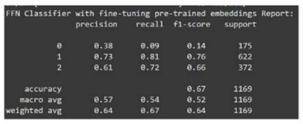
This model was as same as FFN Classifier with pre-trained embeddings but allowed pre-trained Word2Vec embeddings to be trainable, enabling fine-tuning during training. This increased the number of trainable parameters to 5.2m, as the embedding layer was no longer frozen. When it comes to results, there was a slight improvement in accuracy (67%) and got a better F1-score for positive class (0.66) and negative class were under performed. Overall Fine-tuning allowed to slight improvement of generalization for minority samples.

FFN Classifier with fine-tuning pre-trained embeddings



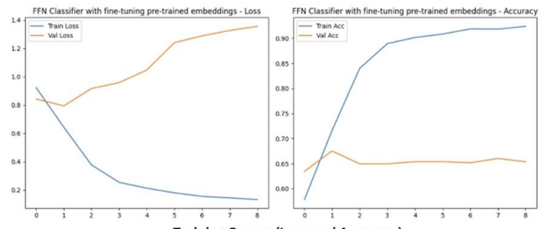
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 50, 300)           3272400

 flatten_1 (Flatten)         (None, 15000)             0

 dense_3 (Dense)             (None, 128)               1920128

 dropout_2 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 dropout_3 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 3)                 195

=================================================================
Total params: 5200979 (19.84 MB)
Trainable params: 5200979 (19.84 MB)
Non-trainable params: 0 (0.00 Byte)
```

Model Summary



```
FFN Classifier with fine-tuning pre-trained embeddings Report:
              precision    recall  f1-score   support

           0       0.38      0.09      0.14       175
           1       0.73      0.81      0.76       622
           2       0.61      0.72      0.66       372

    accuracy                           0.67      1169
   macro avg       0.57      0.54      0.52      1169
weighted avg       0.64      0.67      0.64      1169
```

Classification Report



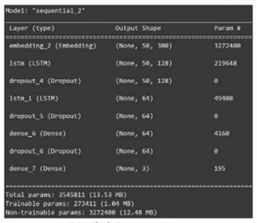
Training Curves (Lose and Accuracy)

FFN Classifier (Frozen) Vs FFN Classifier with Fine-Tuning:

- Both models used the same FFN architecture with two dense layers and dropout and only the embedding layer differed in training process.
- The FFN Classifier (Frozen) used frozen Word2Vec embeddings, while the other model is fine-tuned them during training.
- Fine-tunned model improved accuracy from 64% (frozen) to 67%.
- The F1-score for the positive class improved from 0.56 to 0.66, showing the better generalization.
- Both models are struggled with negative class (F1-score: 0.22 Vs 0.14), likely due to class imbalance.

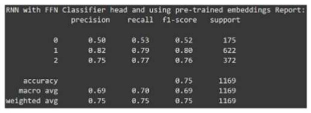
3. RNN with FFN Classifier head and using pre-trained embeddings:

RNN with FFN Classifier head and using pre-trained embeddings



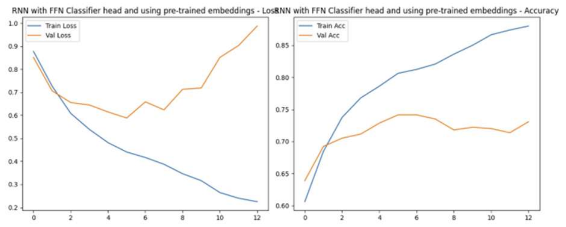
```
Model: "sequential_2"
────────────────────────────────────────────────────────────────
 Layer (type)                  Output Shape              Param #
================================================================
 embedding_2 (Embedding)       (None, 50, 300)           3272400

 lstm (LSTM)                   (None, 50, 128)           219648

 dropout_4 (Dropout)           (None, 50, 128)           0

 lstm_1 (LSTM)                 (None, 64)                49408

 dropout_5 (Dropout)           (None, 64)                0

 dense_6 (Dense)               (None, 64)                4160

 dropout_6 (Dropout)           (None, 64)                0

 dense_7 (Dense)               (None, 3)                 195

================================================================
Total params: 3545811 (13.53 MB)
Trainable params: 273411 (1.04 MB)
Non-trainable params: 3272400 (12.48 MB)
```

Model Summary



```
RNN with FFN Classifier head and using pre-trained embeddings Report:
              precision    recall  f1-score   support

           0       0.50      0.53      0.52       175
           1       0.82      0.79      0.80       622
           2       0.75      0.77      0.76       372

    accuracy                           0.75      1169
   macro avg       0.69      0.70      0.69      1169
weighted avg       0.75      0.75      0.75      1169
```

Classification Report



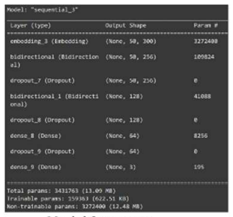
Training Curves (Lose and Accuracy)

This model used a stack of two LSTM layers (128 and 64 units) with dropout for regularization, followed by dense classification head. Pretrained Word2Vec embeddings were kept frozen, and the best configuration – selected through trail and error – achieved 75% accuracy, with strong performance across all classes, including F1-score of 0.52 (negative), 0.80 (neutral), and 0.76 (positive). This architecture significantly outperformed the FFNs, especially on the minority class, due to LSTM's ability to capture sequential context and long-term dependencies. This model had 3.5M total parameters out of those 1.04M used in training, which results in strong generalization.

4. Complex RNN with FFN Classifier head using pre-trained embeddings:

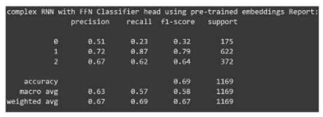
This model used stacked Bidirectional SimpleRNN layers (128 and 64 units) with dropout and a dense classification head, leveraging frozen Word2Vec embeddings. This architecture achieved 69% accuracy, with an F1-score of 0.32 (negative), 0.79 (neutral), and 0.64 (positive). While it captured some sequential context, its performance was slightly lower than the LSTM-based RNN model, particularly on the negative class. The model had 3.43M total parameters, with 159K trainable, and showed decent generalization.

complex RNN with FFN Classifier head using pre-trained embeddings



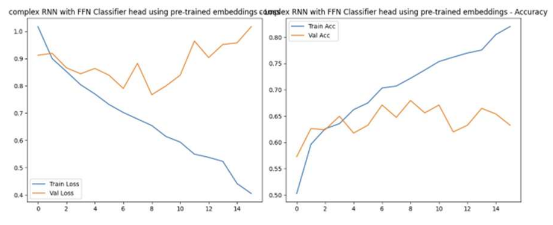
```
Model: "sequential_3"
────────────────────────────────────────────────────────────────
 Layer (type)                  Output Shape              Param #
================================================================
 embedding_3 (Embedding)       (None, 50, 300)           3272400

 bidirectional (Bidirection    (None, 50, 256)           109824
 al)

 dropout_7 (Dropout)           (None, 50, 256)           0

 bidirectional_1 (Bidirecti    (None, 128)               41088
 onal)

 dropout_8 (Dropout)           (None, 128)               0

 dense_8 (Dense)               (None, 64)                8256

 dropout_9 (Dropout)           (None, 64)                0

 dense_9 (Dense)               (None, 3)                 195

================================================================
Total params: 3431763 (13.09 MB)
Trainable params: 159363 (622.51 KB)
Non-trainable params: 3272400 (12.48 MB)
```

Model Summary



```
complex RNN with FFN Classifier head using pre-trained embeddings Report:
              precision    recall  f1-score   support

           0       0.51      0.23      0.32       175
           1       0.72      0.87      0.79       622
           2       0.67      0.62      0.64       372

    accuracy                           0.69      1169
   macro avg       0.63      0.57      0.58      1169
weighted avg       0.67      0.69      0.67      1169
```

Classification Report



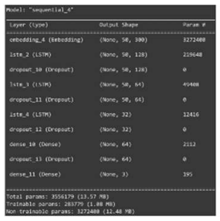
Training Curves (Lose and Accuracy)

RNN with FFN classifier head Vs Complex RNN with FFN Classifier head

- ➢ Model 3 (LSTM) achieved 75% accuracy, outperforming Model 4 (SimpleRNN) which reached 69% accuracy.
- ➢ LSTM showed stronger F1-scores across all classes – 0.52 (negative) vs 0.32, 0.80 (neutral) vs 0.79, and 0.76 (positive) vs 0.64.
- ➢ Model 3 had a macro F1-score of 0.69, while Model 4 lagged behind at 0.58, indicating better balanced performance.
- ➢ Model 3 had more trainable parameters (1.04M) compared to Model 4 (159K), but used the capacity more effectively.

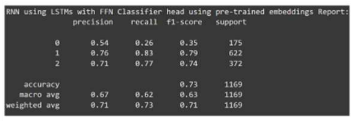
5. RNN using LSTMs with FFN Classifier head using pre-trained embeddings:

RNN using LSTMs with FFN Classifier head using pre-trained embeddings



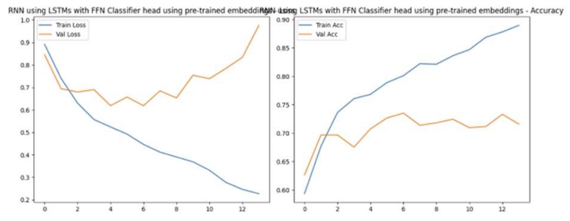
```
Model: "sequential_4"
______________________________________________________________
 Layer (type)                Output Shape              Param #
==============================================================
 embedding_4 (Embedding)     (None, 50, 300)           3272400

 lstm_2 (LSTM)               (None, 50, 128)           219648

 dropout_10 (Dropout)        (None, 50, 128)           0

 lstm_3 (LSTM)               (None, 50, 64)            49408

 dropout_11 (Dropout)        (None, 50, 64)            0

 lstm_4 (LSTM)               (None, 32)                12416

 dropout_12 (Dropout)        (None, 32)                0

 dense_10 (Dense)            (None, 64)                2112

 dropout_13 (Dropout)        (None, 64)                0

 dense_11 (Dense)            (None, 3)                 195

==============================================================
Total params: 3556179 (13.57 MB)
Trainable params: 283779 (1.08 MB)
Non-trainable params: 3272400 (12.48 MB)
```

Model Summary



```
RNN using LSTMs with FFN Classifier head using pre-trained embeddings Report:
              precision    recall  f1-score   support

           0       0.54      0.26      0.35       175
           1       0.76      0.83      0.79       622
           2       0.71      0.77      0.74       372

    accuracy                           0.73      1169
   macro avg       0.67      0.62      0.63      1169
weighted avg       0.71      0.73      0.71      1169
```

Classification Report



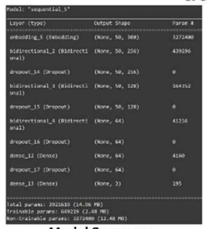
Training Curves (Lose and Accuracy)

The stacked LSTM-based RNN model used three sequential LSTM layers (128, 64 and 32 units) with dropout at each level, followed by a dense classification head, and pretrained Word2Vec embeddings kept frozen. This architecture achieved 73% accuracy, with improved performance across all classes, especially positive (F1 Score: 0.74) and neutral (F1 Score: 0.79), and modest gains for the negative class (F1 Score: 0.35). With ~1.08M trainable parameters, the model balanced complexity and performance well, benefiting from deep temporary representations and high-quality context capture compared to Simpler RNN configurations.

6. RNN using Bi-LSTMs with FFN Classifier head using pre-trained embeddings:

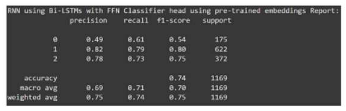
The Bi-LSTM model used three stacked Bidirectional LSTM layer (128, 64, and 32 units) with dropout and a dense classifier, leveraging frozen Word2Vec embeddings. It achieved a strong accuracy of 74%, with balanced performance across all classes – particularly an improved F1-score of 0.54 for the negative class, and high scores for neutral (0.80) and positive (0.75). With ~649K trainable parameters, this model effectively captured both forward and backward temporary dependencies, offering high quality understanding.

RNN using Bi-LSTMs with FFN Classifier head using pre-trained embeddings



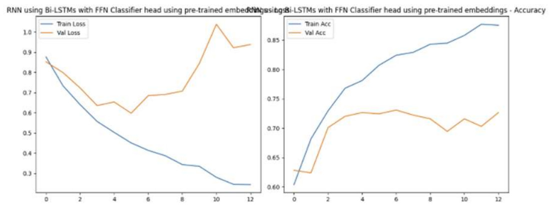
```
Model: "sequential_5"
______________________________________________________________
 Layer (type)                Output Shape              Param #
==============================================================
 embedding_5 (Embedding)     (None, 50, 300)           3272400

 bidirectional_2 (Bidirecti  (None, 50, 256)           439296
 onal)

 dropout_14 (Dropout)        (None, 50, 256)           0

 bidirectional_3 (Bidirecti  (None, 50, 128)           164352
 onal)

 dropout_15 (Dropout)        (None, 50, 128)           0

 bidirectional_4 (Bidirecti  (None, 64)                41216
 onal)

 dropout_16 (Dropout)        (None, 64)                0

 dense_12 (Dense)            (None, 64)                4160

 dropout_17 (Dropout)        (None, 64)                0

 dense_13 (Dense)            (None, 3)                 195

==============================================================
Total params: 3921619 (14.96 MB)
Trainable params: 649219 (2.48 MB)
Non-trainable params: 3272400 (12.48 MB)
```

Model Summary



```
RNN using Bi-LSTMs with FFN Classifier head using pre-trained embeddings Report:
              precision    recall  f1-score   support

           0       0.49      0.61      0.54       175
           1       0.82      0.79      0.80       622
           2       0.78      0.73      0.75       372

    accuracy                           0.74      1169
   macro avg       0.69      0.71      0.70      1169
weighted avg       0.75      0.74      0.75      1169
```

Classification Report



Training Curves (Lose and Accuracy)

RNN using LSTMs with FFN Classifier Vs RNN using Bi-LSTMs with Classifier:

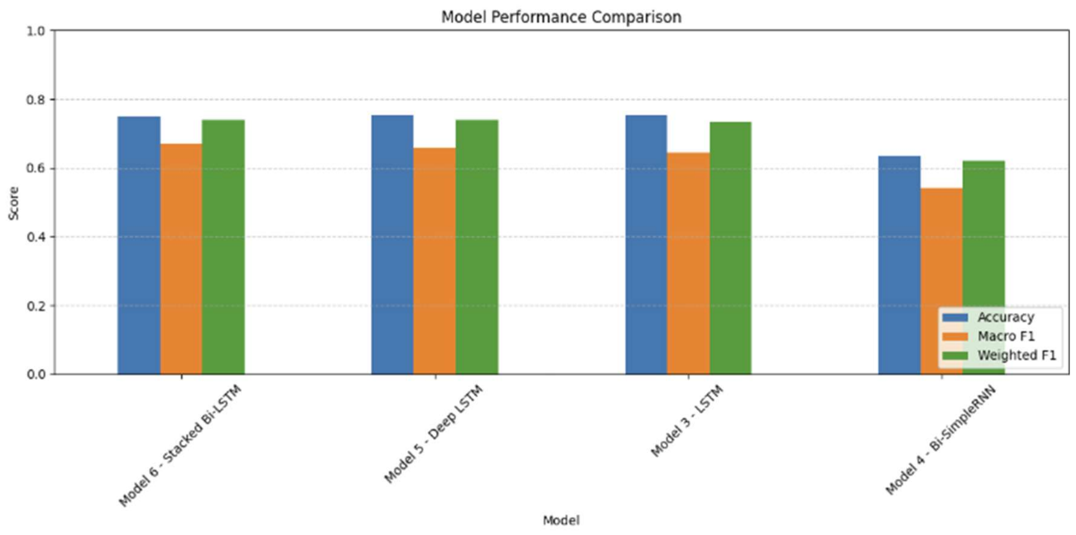
- Model5 (LSTM) achieved 73% accuracy, while Model6 (Bi-LSTM) slightly outperformed it with 74% accuracy.
- Bi-LSTM showed stronger F1-Score – 0.54 (negative) Vs 0.35, 0.80 (neutral) Vs 0.79, and 0.75 (positive) Vs 0.74 – especially improving minority class performance.
- Model 6 achieved a higher macro F1-score of 0.70, compared to 0.63 from Model 5, indicating better balance across all classes.
- Despite higher performance, Bi-LSTM had fewer trainable parameter (649k) compared to 1.08M in LSTM, making it both efficient and effective.

Conclusion on Models 3, 4, 5, and 6:

RNN architectures using pretrained Word2Vec embeddings to capture temporary dependencies:

- **Model 3** (Stacked LSTM) showed strong overall performance with 75% accuracy and balanced F1-scores across classes.
- **Model 4** (Stacked Bidirectional SimpleRNN), while efficient, underperformed with 69% accuracy and weak handling of the negative class, due to the limited memory capability of SimpleRNNs.
- **Model 5** (Deep Stacked LSTM) added an extra LSTM layer for depth, achieving 73% accuracy, but showed limited gains compared to Model 3 and still struggled with minority class prediction.
- **Model 6** (Stacked Bi-LSTM) offered the best trade-off, with 74% accuracy, high F1-scores across all classes, and resulted better handling of the negative class (F1 = 0.54), despite having fewer trainable parameters than Model 5.

Model	Accuracy	Macro F1	Weighted F1
Model 3 – LSTM	0.7536	0.6429	0.7333
Model 4 – Bi-SimpleRNN	0.6330	0.5421	0.6201
Model 5 -Deep LSTM	0.7528	0.6571	0.7386
Model 6 – Stacked Bi-LSTM	0.7494	0.6706	0.7393



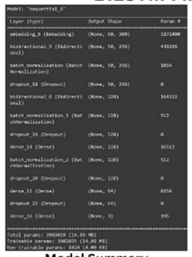
Concluded: MODEL 6 (Stacked Bi-LSTM) is Best Overall Performer.

This model achieved strong, balanced performance, leveraged both way contexts and efficiently captured sequence patterns with a relatively low parameter count – making it most effective and robust architecture.

7. BiLSTM Fine-tuned Embedding Classifier:

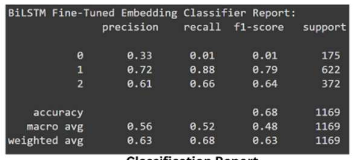
This model has hybrid architecture inspired by the strengths of previous models, particularly Model 6 (stacked Bi-LSTM) and Model 2 (fine-tuned embeddings). This model combines fine-tuned pretrained Word2Vec embeddings, stacked Bidirectional LSTMs, and batch normalization with deep, regularized classification head. The objective is the leverage strong temporary modeling, trainable semantic representations, and enhanced training stability.

BiLSTM Fine-Tuned Embedding Classifier



```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, 50, 300)           3272400

 bidirectional_5 (Bidirecti  (None, 50, 256)           439296
 onal)

 batch_normalization (Batch  (None, 50, 256)           1024
 Normalization)

 dropout_18 (Dropout)        (None, 50, 256)           0

 bidirectional_6 (Bidirecti  (None, 128)               164352
 onal)

 batch_normalization_1 (Bat  (None, 128)               512
 chNormalization)

 dropout_19 (Dropout)        (None, 128)               0

 dense_14 (Dense)            (None, 128)               16512

 batch_normalization_2 (Bat  (None, 128)               512
 chNormalization)

 dropout_20 (Dropout)        (None, 128)               0

 dense_15 (Dense)            (None, 64)                8256

 dropout_21 (Dropout)        (None, 64)                0

 dense_16 (Dense)            (None, 3)                 195

=================================================================
Total params: 3909859 (14.89 MB)
Trainable params: 3902835 (14.89 MB)
Non-trainable params: 1024 (4.00 KB)
```

Model Summary



```
BiLSTM Fine-Tuned Embedding Classifier Report:
              precision    recall  f1-score   support

           0       0.33      0.01      0.01       175
           1       0.72      0.88      0.79       622
           2       0.61      0.66      0.64       372

    accuracy                           0.68      1169
   macro avg       0.56      0.52      0.48      1169
weighted avg       0.63      0.68      0.63      1169
```

Classification Report

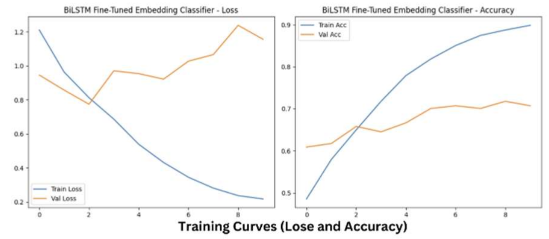
Architecture:

➢ **Embedding Layer:** Initialized with pretrained Word2Vec embeddings but set as trainable to adapt to representation to financial sentiment. This allows the model to refine semantic understanding during the training process.

➢ **Stacked Bidirectional LSTM Layers:** Two Bi-LSTM layers (128 and 64 units) capture both forward and backward dependencies in the text, helping the model understand what the context-sensitive financial phrases are?

➢ **Batch Normalization & Dropout:** Applied after each recurrent and dense layer to stabilize training, speed up convergence, and reduce overfitting. This was crucial since the model has 3.9M total parameters, most of which are now trainable due to fine-tuning.

➢ **Dense Classifier Head:** Two fully connected layers (128 and 64 units) followed by dropout and ReLU activation, enabling non-linear decision boundaries and deeper feature extraction before final classification.



Training Curves (Lose and Accuracy)

Insights:

- The model successfully captured rich semantic and sequential information, performing well on neutral and positive classes.
- However, overfitting and severe class imbalance led to very poor generalization on the negative class, despite deeper architecture and fine-tuning.
- High parameter count introduced optimization complexity, indicating that further regularization or data augmentation might be necessary for improvements.
- Compared to Model 6, Model 7 traded balanced performance for capacity and trainability, but didn't achieve the expected boost due to underrepresented classes and potential overfitting.

Questions:

1. Why did the sentence separator token had to be the first token in the vocabulary dict that you created?

The sentence separator token </s> was placed first in the vocabulary (index 0) to serve as the padding token during sequence processing. Padding tokens are typically assigned index 0. So, models can easily distinguish them from real words and ignore them during training. Especially, in embedding layers and sequence-based models like RNNs.

2. Check to see if your vocabulary has repeated words with different cases (e.g The and the). Why do we have them both? Find 4 more examples of this and report here.

I have kept both forms to preserve the semantic and contextual meaning of casing in financial text – capitalization can indicate emphasis, acronyms or proper nouns. Eg:

['the', 'The', 'THE']

['in', 'In', 'IN']

['for', 'For', 'FOR']

['EUR', 'eur']

['s', 'S']

3. Check Word2Vec embeddings. Are word embeddings (from Word2Vec) similar for these words or are they different? Why?

Word2Vec embeddings are different for case-sensitive words like "Bank" and "bank" because Word2Vec treats them as distinct tokens based on their casing. Where capitalization implies different meaning – eg: "US" is a country while "us" refers to pronoun.

4. Briefly explain what class imbalance is and what problems it might pose. Did you have class imbalance in this assignment? If yes, how did you handle it?

Class imbalance occurs when the number of samples in one or more classes is significantly higher than in other classes. This results model to become biased towards the majority class, while poor performance on minority classes. Yes, This assignment had class imbalance. To handle this, I have used class_weight in Keras during model training. This technique assigned higher weights to underrepresented classes. Improving its ability to generalize across all sentiment categories.