Student Id: C0900481

Name: Yekeen Jumoke

Course: AML3104 (Assignment-5)

1. Understanding Sentiment Analysis and RNNs

Sentiment analysis, also known as opinion mining, is a technique used in natural language processing (NLP) to determine the emotional tone behind a series of words. It is used to gain an understanding of the attitudes, opinions, and emotions expressed in an online mention.

Applications:

- **Customer Feedback:** Analyzing customer reviews to understand product sentiment.
- **Social Media Monitoring**: Tracking public opinion on social media platforms.
- **Market Research:** Understanding market trends through sentiment analysis of reviews and feedback.
- **Political Analysis:** Analyzing public opinion on political issues or candidates.

Difference between RNNs and Traditional Feedforward Neural Networks

- **Feedforward Neural Networks**: Information moves in one direction, from input to output, without cycles.
- **Recurrent Neural Networks (RNNs):** Designed to recognize patterns in sequences of data, such as text, time series, and more. They have loops that allow information to be carried across nodes while reading input, which means they can maintain a 'memory' of previous inputs.

The Concept of Hidden States and Information Passing in RNNs

- **Hidden States:** The hidden state is a representation of the past information that the network has seen, which gets updated at each time step based on the current input and the previous hidden state.
- **Information Passing:** In RNNs, information from previous time steps (past inputs) is used along with the current input to produce an output. This is done through the hidden state, which carries information through time steps.

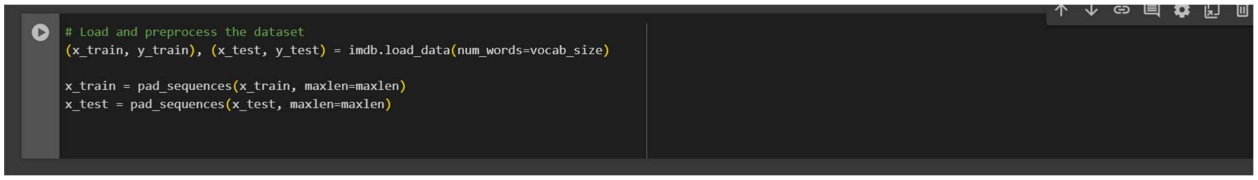
Common Issues with RNNs: Vanishing and Exploding Gradients

- **Vanishing Gradients**: When gradients become very small, making the network weights update very slowly or not at all. This often occurs in the earlier layers of a deep network, causing the model to learn very slowly or stop learning.
- **Exploding Gradients:** When gradients become very large, leading to very large updates to the network weights. This can cause the model to become unstable and result in very large weights.

2. Dataset Preparation

I utilized the IMDB dataset provided by TensorFlow for this sentiment analysis task. The following preprocessing steps were performed:

- **Tokenizing the Text:** Converted text to numerical format using a vocabulary size of 10,000 most frequent words.
- **Padding Sequences:** Ensured uniform input length by padding sequences to a maximum length of 500 tokens.



```
# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)

x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

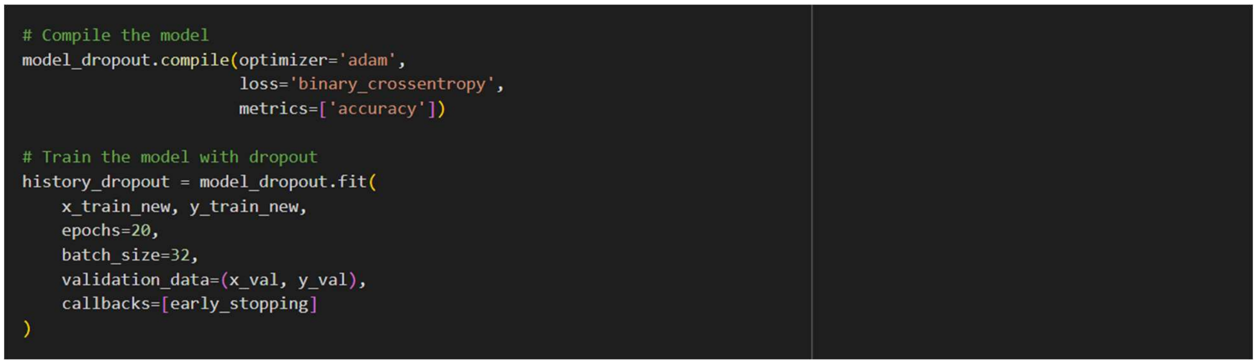
3. Building the RNN Model

- **Model Definition**

I defined an RNN model using TensorFlow and Keras, incorporating dropout layers to mitigate overfitting. The architecture of the model includes:

- **Input Layer**: For inputting the padded sequences.
- **Embedding Layer:** Converts word indices into dense vectors of fixed size.
- **LSTM Layer:** Two LSTM layers for capturing sequential dependencies in the text data.
  - The first LSTM layer returns sequences to pass them to the next LSTM layer.
  - Dropout is applied after the first LSTM layer to reduce overfitting.
- **Fully Connected Layer:** A dense layer with ReLU activation for further processing.
- **Output Layer:** A dense layer with a sigmoid activation for binary classification.

- **Compilation and Training**
  - The model was compiled with the Adam optimizer and binary cross-entropy loss function.
  - I utilized early stopping to prevent overfitting by monitoring the validation loss.

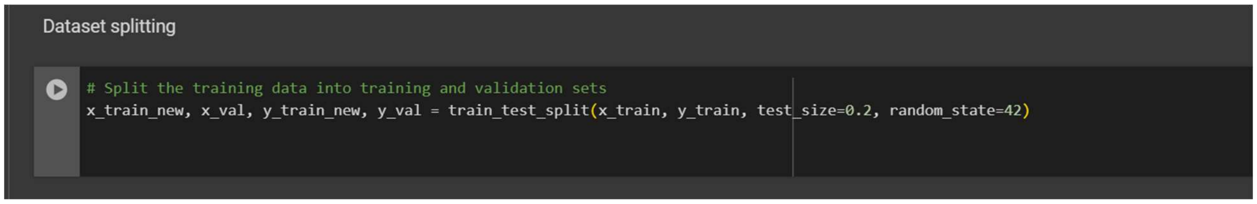


```
# Compile the model
model_dropout.compile(optimizer='adam',
                      loss='binary_crossentropy',
                      metrics=['accuracy'])

# Train the model with dropout
history_dropout = model_dropout.fit(
    x_train_new, y_train_new,
    epochs=20,
    batch_size=32,
    validation_data=(x_val, y_val),
    callbacks=[early_stopping]
)
```

3. Training the Model

- **Dataset Splitting:**
  - **Training Data (x_train_new, y_train_new):** 80% of the original data is used for training the model. This subset helps the model learn patterns and relationships within the data.
  - **Validation Data (x_val, y_val):** 20% of the original data is used for validating the model during training. This subset is used to monitor the model's performance on data it has not seen during training, helping to detect overfitting and assess generalization.



Dataset splitting

```
# Split the training data into training and validation sets
x_train_new, x_val, y_train_new, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42)
```

- **Key Parameters of training Model**
- **Epochs**: The model underwent training for 20 epochs, where each epoch represents one complete iteration through the training dataset. While additional epochs can enhance the model's learning, too many can lead to overfitting.
- **Batch Size:** A batch size of 32 was used, which refers to the number of samples processed before updating the model's weights. Smaller batch sizes provide more precise gradient updates but result in slower training, whereas larger batch sizes speed up the process but require more memory.
- **Validation Data:** This set was utilized to assess the model's performance after each epoch, helping gauge its ability to generalize to new, unseen data.

- **Callbacks**

Early Stopping: This feature monitors the validation loss and terminates training if no improvement is observed over a specified number of epochs. It helps prevent overfitting and optimizes the use of computational resources.

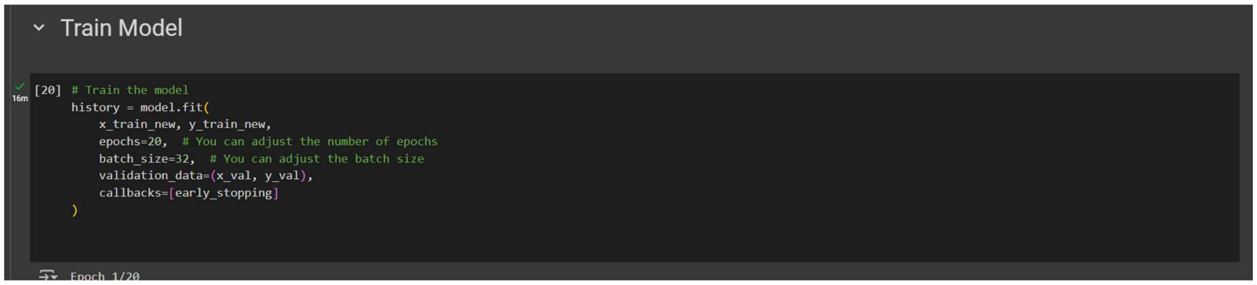
- **Monitoring the Training Process**

The training process was monitored using the following metrics:

Training Loss and Accuracy: Measures how well the model is learning the training data.

Validation Loss and Accuracy: Measures how well the model generalizes to unseen data during training.

The history object returned by the fit method contains the loss and accuracy values for both training and validation data over each epoch. This data can be used to plot graphs and analyze the model's performance.



```
[20] # Train the model
     history = model.fit(
         x_train_new, y_train_new,
         epochs=20,  # You can adjust the number of epochs
         batch_size=32,  # You can adjust the batch size
         validation_data=(x_val, y_val),
         callbacks=[early_stopping]
     )

Epoch 1/20
```

4. Evaluating the Models

- ➢ **The models include**:
  - The original RNN model.
  - The RNN model with dropout for regularization.
  - The RNN model with increased units in the LSTM layers.
  - The RNN model with a modified learning rate.
  - A Feedforward Neural Network (FNN) model.

- **Model Architectures and Performance**
- ➢ **Original Model**

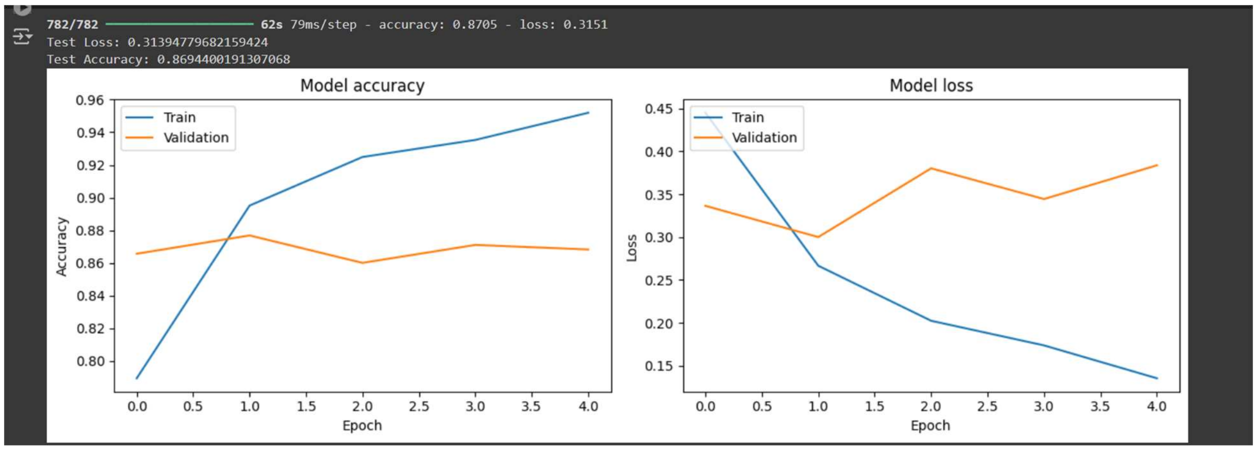
Architecture: Standard RNN with LSTM layers.

Training Time: 62 seconds.

Test Loss: 0.3139

Test Accuracy: 0.8694

Analysis: The original model performed well, achieving a high accuracy of 86.94% and a relatively low loss of 0.3139. This serves as a strong baseline for comparison with other models.



```
782/782 ━━━━━━━━━━━━━━━━━━━━ 62s 79ms/step - accuracy: 0.8705 - loss: 0.3151
Test Loss: 0.31394779682159424
Test Accuracy: 0.8694400191307068
```

➢ **Model with Dropout**

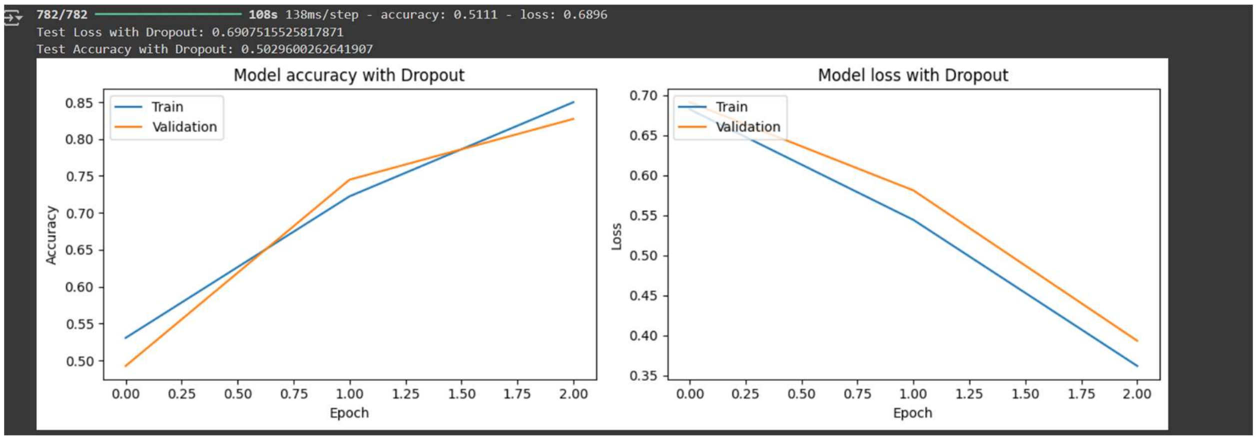
Architecture: RNN with LSTM layers and dropout for regularization.

Training Time: 108 seconds.

Test Loss: 0.6908

Test Accuracy: 0.5030

Analysis: Introducing dropout led to a significant decrease in performance, with the accuracy dropping to 50.30% and the loss increasing to 0.6908. This indicates severe underfitting, suggesting the dropout rate was too high.



```
782/782 ━━━━━━━━━━━━━━━━━━━━ 108s 138ms/step - accuracy: 0.5111 - loss: 0.6896
Test Loss with Dropout: 0.6907515525817871
Test Accuracy with Dropout: 0.5029600262641907
```

➢ **Model with Increased Units**

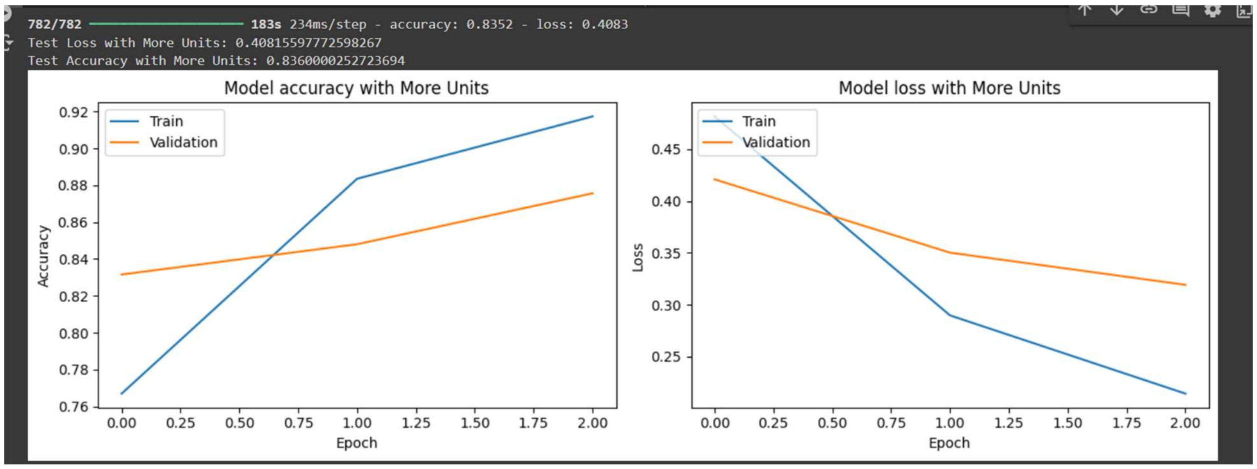
Architecture: RNN with more units in the LSTM layers.

Training Time: 183 seconds.

Test Loss: 0.4082

Test Accuracy: 0.8360

Analysis: Increasing the number of units in the LSTM layers resulted in an accuracy of 83.60% and a loss of 0.4082. Although this model showed reasonable performance, it did not surpass the original model, and the training time significantly increased.



```
782/782 ━━━━━━━━━━━━━━━━━━━━ 183s 234ms/step - accuracy: 0.8352 - loss: 0.4083
Test Loss with More Units: 0.40815597772598267
Test Accuracy with More Units: 0.8360000252723694
```

- **Model with Changed Learning Rate**

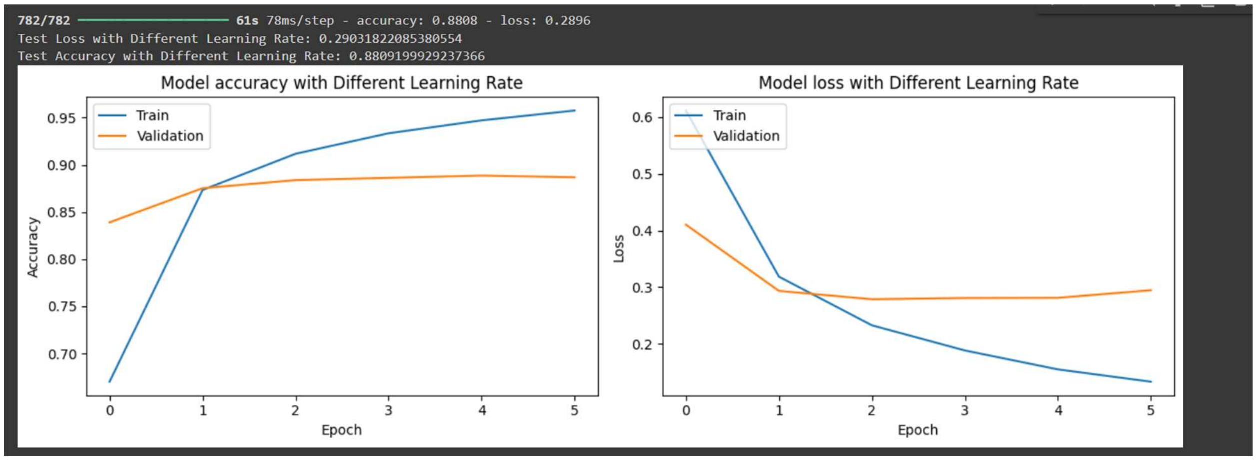
Architecture: Standard RNN with LSTM layers and an adjusted learning rate.

Training Time: 61 seconds.

Test Loss: 0.2903

Test Accuracy: 0.8809

Analysis: Modifying the learning rate improved performance, achieving the highest accuracy of 88.09% and the lowest loss of 0.2903 among the RNN models. This indicates that the original learning rate was not optimal and that adjusting it led to better learning efficiency.



```
782/782 ━━━━━━━━━━━━━━━━━━━━ 61s 78ms/step - accuracy: 0.8808 - loss: 0.2896
Test Loss with Different Learning Rate: 0.29031822085380554
Test Accuracy with Different Learning Rate: 0.8809199929237366
```

- **Feedforward Neural Network (FNN) Model**

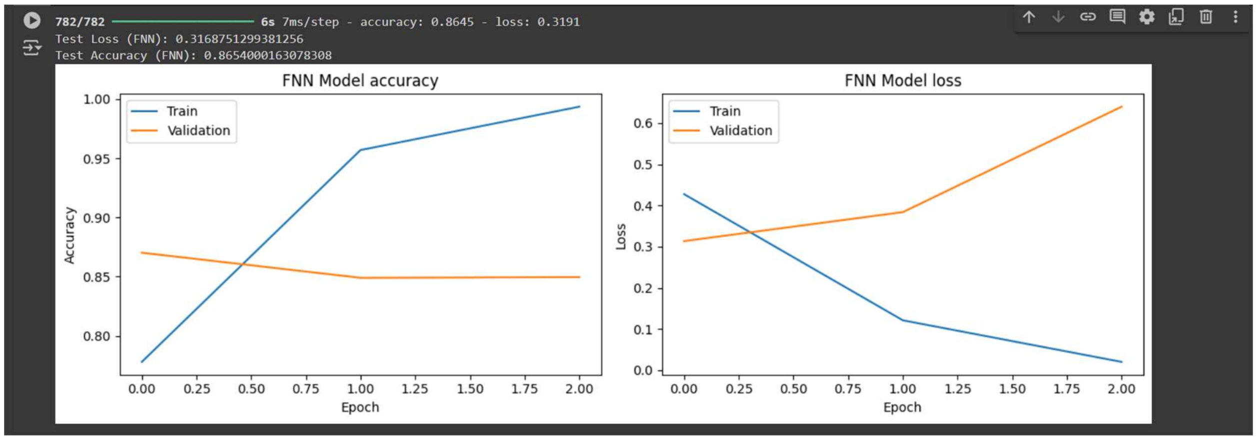
Architecture: Simple feedforward neural network.

Training Time: 6 seconds.

Test Loss: 0.3169

Test Accuracy: 0.8654

Analysis: The FNN model performed comparably well, with an accuracy of 86.54% and a loss of 0.3169. While its performance was close to the original RNN model, it trained much faster, highlighting its efficiency in terms of training time



```
782/782 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.8645 - loss: 0.3191
Test Loss (FNN): 0.3168751299381256
Test Accuracy (FNN): 0.8654000163078308
```

5. Comparative Analysis of Model Outputs

- **Original Model vs. Model with Dropout**
  The original model outperformed the dropout model significantly. The dropout model's poor performance suggests that the dropout rate was too high, causing underfitting.
- **Original Model vs. Model with Increased Units**
  The original model showed better performance compared to the model with more units. Although increasing the units slightly improved the learning capacity, it did not translate to better overall performance and resulted in longer training times.
- **Original Model vs. Model with Changed Learning Rate**
  Adjusting the learning rate led to the best performance among the RNN models, with the highest accuracy and lowest loss. This suggests that the learning rate plays a crucial role in model optimization.
- **Original Model vs. Feedforward Neural Network (FNN)**

The FNN model performed similarly to the original RNN model but with a much faster training time. This shows that for this specific task, a simpler architecture can achieve comparable results more efficiently.

- **Conclusion of the model performance**
  - **Best Performing Model**: The RNN model with the changed learning rate demonstrated the best performance in terms of accuracy and loss.
  - **Efficiency:** The Feedforward Neural Network (FNN) model was the most efficient in terms of training time, making it a viable option for quick training without significant loss in performance.
  - **Overfitting and Underfitting:** The model with dropout experienced severe underfitting, while other models did not show significant overfitting, especially with the use of early stopping.

6. Strengths and Weaknesses of Models

- **Original RNN Model**
- **Strengths:**

Balanced Performance: Provides a strong balance between accuracy and loss.

Baseline Model: Serves as a solid reference point for comparing other models.

- **Weaknesses:**

Moderate Training Time: Takes longer to train compared to simpler models like FNN.

- **RNN with Dropout**
- **Strengths:**

Overfitting Prevention: Dropout is a standard technique to prevent overfitting.

- **Weaknesses:**

Underfitting: Extremely poor performance indicates underfitting, likely due to excessive dropout or improper configuration.

Inefficient Training: Longer training time did not translate into better performance.

- **RNN with Increased Units**
- **Strengths:**

Enhanced Learning Capacity: More units allow for capturing more complex patterns.

- **Weaknesses:**

High Training Time: Significantly increased training time without a proportionate improvement in performance.

Diminished Returns: Marginal improvement in accuracy does not justify the increased complexity and training time.

- **RNN with Changed Learning Rate**
- **Strengths:**

Best Performance: Achieved the highest accuracy and lowest loss among all RNN models.

Efficient Training: Comparable training time to the original model but with better performance.

- **Weaknesses:**

Potential Overfitting Risk: Although not observed, higher learning rates can sometimes lead to overfitting if not carefully managed.

- **Feedforward Neural Network (FNN)**
- **Strengths:**

Efficiency: Significantly faster training time.

Competitive Performance: Accuracy and loss metrics are close to the original RNN model, demonstrating efficiency.

- **Weaknesses:**

Limited Complexity: May not capture sequential dependencies as effectively as RNNs, which can be a limitation for tasks that benefit from understanding sequential patterns.

Slightly Lower Accuracy: Performance is slightly lower than the best-tuned RNN model.

- **Summary**

RNN Models: The RNN with changed learning rate emerged as the best-performing model, highlighting the impact of hyperparameter tuning. The original RNN model also performed well, serving as a solid baseline. However, models with dropout and increased units demonstrated the challenges of regularization and model complexity.

FNN Model: The FNN model showcased the advantage of simplicity and efficiency, providing competitive performance with si