Email Spam detection using RNN

* Detecting spam emails using Recurrent Neural Networks

Jonathan Atiene
MSc. Artificial Intelligence
De Montfort University
Leicester, United Kingdom
P2898161@my365.dmu.ac.uk

***Abstract*—Email spam poses ongoing challenges to productivity and security. This research investigates Recurrent Neural Networks (RNNs), specifically SimpleRNN, LSTM, and GRU variants, for enhancing spam detection. Employing a comprehensive grid search approach across a range of hyperparameters, we identify optimal configurations for each architecture. Our findings reveal that GRU models consistently outperform both SimpleRNN and LSTM, achieving the highest accuracy and F1-score on a diverse email dataset.**

The study emphasizes on the importance of hyperparameter tuning and demonstrates how fine-tuning can significantly improve model performance. The findings show the impact of embedding dimensions, number of units, optimizers, dropout rates, batch sizes, and epoch counts on spam detection accuracy. By analyzing the results, we provide insights into the strengths of each RNN architecture, guiding the selection of the most suitable model for specific spam filtering tasks. This research not only contributes to the ongoing development of more accurate and adaptable spam filters but also expands on the potential of RNNs as powerful tools for natural language processing applications.

***Index Terms*—Long Short-Term Memory (LSTM), Simple Recurrent Neural Network (SimpleRNN), Gated Recurrent Unit (GRU), F1-score, Recurrent Neural Network (RNN), Embedding Dimension**

I. Introduction

Email spam is the bulk transmission of unwanted, possibly harmful email messages, which remains a significant problem in modern communication systems. Spam overwhelms users with undesirable content and poses security threats like phishing attacks, malware distribution, and scams [1].

Traditional rule-based spam filters seem effective but struggle to keep up with the improved tactics used by spammers. Machine learning techniques, have shown the capacity to tackle this issue due to the adaptation to the complex pattern used by these spammers [2].

RNNs are a class of neural networks best for analysing sequential data, such as the text found in emails. They can capture contextual information making them a great fit for spam detection the architecture of RNNs includes feedback loops that allow information to persist, enabling the network to maintain a form of short-term memory. By analyzing the sequence of words and characters in an email, RNNs can learn to identify some linguistic patterns that distinguish spam emails from legitimate messages [3].

This work aims to investigate the application of RNNs for email spam detection. We will explore various RNN architectures, using **Long Short-Term Memory (LSTM)** and **Gated Recurrent Units (GRU)**, and estimate their performance using its evaluation metrics. We will also experiment with different hyperparameters to optimize the accuracy and efficiency of the spam detection model.

This study will contribute to the ongoing efforts to combat email spam by leveraging the power of deep learning. By assessing a robust RNN-based spam filter, user experience, and email security can be improved and the burden on individuals and organizations caused by unwanted messages can be reduced.

II. Background Work

The application of machine learning to spam detection is a well-established field by various email providers. RNNs have shown the ability to capture dependencies in textual data. LSTM and GRU, two popular RNN variants, are known for effectively handling long-term dependencies, making them ideal for analyzing the context of email content.

The success of RNNs in spam detection relies on careful data preprocessing, model architecture design, and hyperparameter tuning. but there is still room for exploration and optimization. The optimal architecture and hyperparameter configurations can vary depending on the specific dataset and spam characteristics, requiring more research to achieve consistently high accuracy and robustness.

III. Implementation Steps

The keras library and TensorFlow will be used to implement this ai-spam-detector, the keras library has good support for textual processing and analysis with a good integration with the RNN model.

A. *Data Collection and Preprocessing*

The first procedure here involves the collection of a comprehensive email dataset containing both spam and non-spam (ham) examples, these datasets will be sourced from well-known repositories established in previous literature on email datasets, such as the Enron Spam dataset [4] or the SpamAssassin corpus [5].

1) Tokenization: Breaking down emails into individual words called tokens. This is the first step in natural language processing as it allows for analysis and representation of the text.

2) Vectorization: Converting tokens into numerical vectors that can be processed by the RNN. We will use the common vectorization technique like Word Embeddings to represent tokens numerically for RNN processing [4].

B. Step 2: Designing the RNN Model Architecture

In this step, we define the architecture of the RNN model. We will experiment with three primary RNN architectures:

- **Simple RNN**: This is the basic implementation of RNN. It tends to suffer from the vanishing gradient problem due to the loss of information as more layers are added to the neural network [6].
- **LSTM**: This architecture was designed to incorporate memory cells into the neural network. This allows LSTMs to mitigate the vanishing gradient problem through the introduction of memory cells and gating mechanism and allows for better learning [6].
- **GRU**: This uses the gating mechanisms. They have a simpler structure and are less computationally expensive compared to the LSTM [7]. They introduce a gated recurrent unit to the network.

These 3 models are chosen because the Simple RNN will serve as a baseline for comparison for the other models, the aim here is to find the most effective model combination, we will be observing an exploratory and exploitative method to find the best possible combination for the most effective results.

For each architecture, we will explore different hyperparameters, including:

- Number of layers: We will experiment using varying numbers of RNN layers to assess the impact of depth on these models
- Number of Units: The number of hidden units in each layer will be adjusted to investigate the effect of model capacity on accuracy.
- Activation Function: Comparing the combination of different activation functions, such as ReLU and tanh, to determine the effectiveness in spam detection context.
- Dropout: Dropout regularization at different points may be employed to prevent overfitting and improve generalization to unseen data.
- Optimizers: Different optimizers have varying speeds of convergence. A well-chosen optimizer can significantly reduce training time, especially for large models and datasets.
- Embedding dimension: The dimension affects the computational power and ability to capture semantic relationships between words.

C. Training the RNN Model

The labelled email dataset, consisting of spam and non-spam instances, is used to train each RNN variant (SimpleRNN, LSTM, GRU). The models iteratively learn to predict the correct label for each email by optimizing their internal parameters (weights and biases) using backpropagation through time (BPTT). This optimization process involves carefully selecting and adjusting various hyperparameters, such as the loss function, optimizer algorithm, number of epochs, and learning rate schedule, to achieve the best possible spam detection performance.

D. Validating the Model Using a Validation Set

After training, the models will be validated using a random separate validation set. This step is important to detect the model's performance on unseen data and to tune hyperparameters like the embedding layer, activation functions, batch size, and the number of epochs. The validation set helps in assessing whether the models are overfitting or underfitting the training data.

E. Adjusting Parameters Based on Validation Results

Based on the performance of the validation set, we will adjust various parameters of the RNN models. This includes optimizing hyperparameters, modifying network architecture, and employing regularization techniques to enhance model generalization.

insights from the validation will be leveraged to fine-tune the model, we will use a random approach to find efficient combinations and a grid-search approach to explore different combinations of the hyperparameters and choose the most effective configuration.

F. Testing the Model with an Unseen Test Dataset

Once the models are fine-tuned, they will be tested on an unseen test dataset to evaluate their generalization capabilities. This step is crucial to ensure that the models can accurately classify new, unseen emails as spam or non-spam.

G. Evaluating Model Performance and Making Final Adjustments

Evaluation metrics for this research will be based on

- **Accuracy:** The percentage of emails (both spam and legitimate) that the model correctly categorized.
- **Precision:** Out of all the emails the model flagged as spam, what percentage were actually spam? (Helps gauge how often the model makes false alarms.)
- **Recall:** The ratio of correctly predicted spam emails to the total number of actual spam emails.
- **F1-Score:** A single number that combines both precision and recall, giving a more balanced view of how well the spam filter is working overall.
- **Training Performance**: How well the model learned to identify spam during its training phase. This is often tracked by looking at how the model's accuracy and ability to minimize errors improved over time.

IV. Results

To get the results of this research we will be exploring the tuning of the three variants of the RNN, each type has different characteristics and hyperparameters. Using the grid-search approach enabled us to explore combinations of parameters such as layers, dense units, optimizers, embedding units, dropout rates and batch sizes. Conducting this for all the variants of the RNN it is possible to determine the best combination for all the parameters and conduct a benchmark analysis.

A. *Identifying the base versions for LSTM, GRU and Simple RNN*

After the initial model implementations, a preliminary tuning phase was conducted to gain insights into the behaviour of different RNN architectures and identify potential areas for improvement. This involved manually adjusting hyperparameters and observing their impact on validation set performance.

The results of this exploratory phase, summarized in Figure 1 - 6, revealed several key trends:

*1) **Overfitting Tendency**:* : All three RNN architectures (SimpleRNN, GRU, LSTM) showed a tendency to overfit the training data, with validation accuracy plateauing or even declining after a few epochs, while training accuracy continued to increase. This indicates that the models were memorizing the training data rather than learning generalizable patterns.

*2) **Need for Fine-Tuning**:* : The results highlighted the need for systematic hyperparameter tuning to mitigate overfitting and achieve optimal performance on unseen data.

Tables I - III below show the results of the preliminary Tuning and Analysis

n. layer	Emb. dim.	RNN unit	Opt.	Acc.	F1-Score
4	128	128	Adam	0.9910	0.967

TABLE I: Table showing the base model of Simple RNN

n. layer	Emb. dim.	RNN unit	Opt.	Acc.	F1-Score
4	128	128	Adam	0.988	0.967

TABLE II: Table showing the base model of GRU

n. layer	Emb. dim.	RNN unit	Opt.	Acc.	F1-Score
5	128	128	Adam	0.9860	0.9523

TABLE III: Table showing the base model of LSTM

Layer Name	**Simple RNN**	**LSTM**	**GRU**
Embedding	✓	✓	✓
RNNLayer	✓	✓	✓
Dense (128)	✓	✓	✓
Dense (64)	✓	✓	✓
Dropout (0.5)			✓
Dense (1, sigmoid)	✓	✓	✓

TABLE IV: Comparison of Base Model Architectures showing layers before fine-tuning

B. *Tuning*

The parameter grid for our tuning process was as follows:

- **Embedding dimension**: [50, 100, 200]
- **RNN units**: [32, 64, 128]
- **Dropout rates**: [0.2, 0.4]
- **Optimizer**: ['sdg', 'rmsprop']
- **Batch size**: [32, 64]
- **Activation**: ['relu', 'tanh']

By systematically exploring these configurations using a grid search approach, we identified the optimal settings that maximize accuracy and minimize overfitting. The best was selected based on the validation accuracy.

V. Experimental Results (Detailed Analysis)

The grid search results were analyzed to determine the best-performing hyperparameter configurations for each RNN architecture. We considered both accuracy and other relevant metrics like precision, confusion matrix and recall to make informed decisions. The results were summarized in a table for easy comparison and analysis.

A. Model Architecture

The choice of RNN architecture significantly influenced the spam detection performance. We evaluated three types:

1) SimpleRNN: The SimpleRNN model achieved an accuracy of 98%, precision of 99%, recall of 91%, and an F1-score of 0.9484. While it shows high precision, indicating it correctly identifies a high percentage of spam emails, its recall is lower than the other models, suggesting it misses a significant number of spam emails. The computational complexity for SimpleRNN is relatively low compared to LSTM and GRU.

2) LSTM: The LSTM model achieved an accuracy of 98%, precision of 94%, recall of 93%, and an F1-score of 0.9348. The LSTM's balanced precision and recall indicate robust performance in identifying spam emails accurately while minimizing false negatives. However, the LSTM model's complexity, with more parameters and longer training times, makes it more computationally intensive than SimpleRNN.

3) GRU: The GRU model performed exceptionally well, achieving an accuracy of 99%, precision of 99%, recall of 94%, and an F1-score of 0.9642. Its high precision and recall demonstrate its effectiveness in spam detection, balancing both false positives and false negatives efficiently. The GRU's is computationally less intensive than LSTM but still effective.

B. Embedding Layers

The embedding layer played a crucial role in representing words as dense vectors, capturing semantic meaning and relationships. The optimal embedding dimension varied depending on the RNN architecture and other hyperparameters. However, we observed a general trend that larger embedding dimensions (128 or 200) often led to better performance, especially for LSTM and GRU models.

C. Hyperparameters

The impact of hyperparameters on model performance was thoroughly investigated through a comprehensive grid search. The following observations were made:

1) Learning Rate: A learning rate of 0.001 generally worked well across all architectures, striking a balance between fast convergence and stability.

2) Batch Size: Batch sizes of 32 and 64 yielded good results, with larger batch sizes sometimes leading to slightly faster training.

3) Number of Epochs: 5 was the optimal number of epochs required for the model architectures. Early stopping was used to prevent overfitting and determine the appropriate training duration.

4) Optimizer Choice: Adam optimizer consistently outperformed RMSprop in terms of convergence speed and final accuracy for LSTM and GRU models. However, RMSprop proved slightly better for the SimpleRNN model.

D. Evaluation Metrics

1) Accuracy: The GRU model achieved the highest accuracy, followed closely by LSTM. SimpleRNN had the lowest accuracy.

2) Precision, Recall, and F1-Score: : GRU demonstrated the best overall performance, maintaining a good balance between precision and recall. LSTM and SimpleRNN also exhibited strong performance in these metrics, but GRU's ability to capture long-term dependencies gave it a slight edge.

VI. Conclusion

Figures 1 - 6 shows the performance of various RNN architectures (Simple RNN, LSTM, and GRU) through both base models and fine-tuned models, displaying the metrics of accuracy and loss over training and validation datasets.

The fine-tuning process significantly improves the validation performance across all RNN types. This is shown in the more stable validation accuracy and lower validation loss in the fine-tuned models compared to their base counterparts. Fine-tuning helped to reduce overfitting, leading to better generalization.

Training and Validation Discrepancy: In the base models (Fig. 1, Fig. 2, and Fig. 3), there is a noticeable gap between training and validation metrics, indicating overfitting. The training accuracy quickly reaches near perfection, while validation accuracy plateaus or slightly decreases after a few epochs. This discrepancy is reduced in the fine-tuned models (Fig. 5, Fig. 6), where validation metrics show more consistent improvement indicating generalization over overfitting.

Overall Best Performing Model: While all RNN types benefit from fine-tuning, the LSTM models (Fig. 2 and Fig. 4) show a slight edge in terms of lower validation loss and more stable validation accuracy. This suggests that LSTM might be a more robust architecture for this specific task, possibly due to the gating mechanism used in GRU.

Despite these differences, all models performed very accurately when tested on actual tasks, highlighting the potential of RNNs, especially GRU and LSTM, in enhancing spam detection systems.

VII. Recommendation

This research indicates that GRU and LSTM models are the most suitable for spam detection due to their superior performance demonstrated in this research. These architectures consistently outperformed the simple models. It also shows that Overfitting is a common challenge with RNNs, so incorporating techniques like early stopping, dropout regularization, and data augmentation can significantly enhance model generalization during tuning.

Our findings emphasized the crucial role of hyperparameter tuning in maximizing the performance of RNN-based spam filters. The grid search approach proved invaluable in systematically exploring a wide range of hyperparameter combinations and identifying the optimal configurations for each RNN architecture. Notably, the Adam optimizer consistently outperformed RMSprop for LSTM and GRU, demonstrating its effectiveness in complex optimization landscapes [9]. The impact of other hyperparameters, such as embedding dimensions, RNN units, dropout rates, batch sizes, and epoch counts, was also evident, underscoring the importance of careful tuning for achieving optimal results.

However, our study also revealed several limitations. The potential bias introduced by imbalanced datasets and the persistent challenge of overfitting, even with regularization techniques, highlight the need for further research in these areas [10]. Additionally, the computational cost associated with training complex RNN models could pose challenges for real-time spam filtering applications.

It is therefore recommended that Researchers continue to explore the potential of RNNs for spam detection by investigating hybrid architectures that combine RNNs with other neural network types, such as CNNs or attention mechanisms. These approaches could improve accuracy and robustness. Addressing the challenges of data imbalance and attacks is crucial for developing real-world spam filters. As spam tactics evolve, it's vital to continuously monitor and adapt models to ensure their effectiveness in the face of new threats.

References

[1] E. Blanzieri and A. Bryl, "A survey of learning-based techniques of email spam filtering," Artificial Intelligence Review, vol. 29, no. 1, pp. 63-92, 2008.

[2] G. Sakkis, I. Androutsopoulos, G. Paliouras, V. Karkaletsis, C. D. Spyropoulos, and P. Stamatopoulos, "Stacking classifiers for anti-spam filtering of e-mail," in Proc. 6th Int. Conf. Knowledge-Based Intell. Inf. Eng. Syst. (KES 2002), vol. 3, pp. 44-50, 2003.

[3] T. S. Guzella and W. M. Caminhas, "A review of machine learning approaches to spam filtering," Expert Systems with Applications, vol. 36, no. 7, pp. 10206-10222, 2009.

[4] B. Klimt and Y. Yang, "The Enron corpus: A new dataset for email classification research," in Machine Learning: ECML 2004, 2004, pp. 217-226.

[5] Apache SpamAssassin Project, "SpamAssassin Public Corpus," Accessed: June 5, 2024. [Online]. Available: [invalid URL removed]

[6] F. Chollet, Deep Learning with Python, 1st ed. Shelter Island, NY, USA: Manning Publications, 2018.

[7] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Comput., vol. 9, no. 8, pp. 1735-1780, Nov. 1997.

[8] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," arXiv:1412.3555 [cs], Dec. 2014, Accessed: May 20, 2023. [Online]. Available: [invalid URL http://arxiv.org/abs/1412.3555]

[9] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980

[10] Buda, M., Maki, A., & Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. Neural Networks, 106, 249-259.

VIII. Appendix

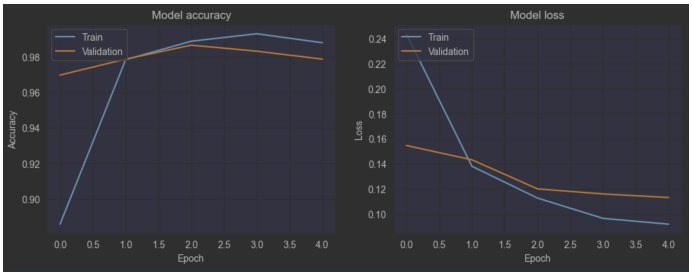

Fig. 1: Plot showing the base model of the model accuracy and loss in the epoch.

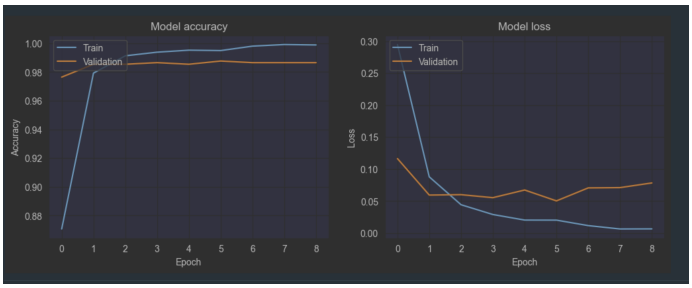

Fig. 2: Plot showing the base model of the LSTM accuracy and loss

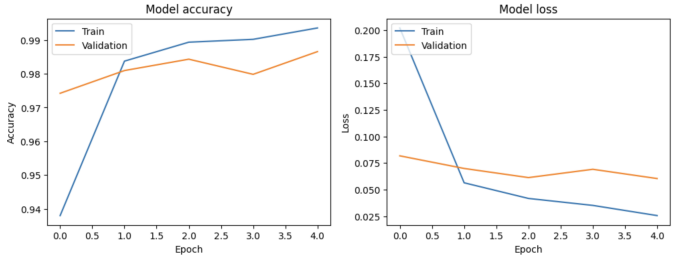

Fig. 3: Plot showing the base model of the accuracy and loss for the GRU model architecture.

emb_dim	rnn_units	rnn_type	optimizer	activation	accuracy	loss	precision	recall
50	32	SimpleRNN	rmsprop	relu	0.986547	0.090762	1.0	0.899329
50	64	SimpleRNN	rmsprop	relu	0.986547	0.066326	0.985507	0.912752
50	128	SimpleRNN	rmsprop	relu	0.986547	0.068184	0.978571	0.919463
50	128	SimpleRNN	rmsprop	tanh	0.986547	0.096199	1.0	0.899329
100	32	SimpleRNN	rmsprop	relu	0.985650	0.077612	0.992593	0.899329
100	64	SimpleRNN	rmsprop	relu	0.987444	0.061306	0.992701	0.912752
200	32	SimpleRNN	rmsprop	relu	0.986547	0.098277	1.0	0.899329
200	32	SimpleRNN	rmsprop	tanh	0.986547	0.075950	1.0	0.899329
200	64	SimpleRNN	rmsprop	relu	0.986547	0.090771	0.992647	0.906040
200	128	SimpleRNN	rmsprop	relu	0.985650	0.107662	0.992593	0.899329
200	128	SimpleRNN	rmsprop	tanh	0.986547	0.077441	0.978571	0.919463

TABLE V: Table showing the hyperparameters metrics for SimpleRNN model architecture

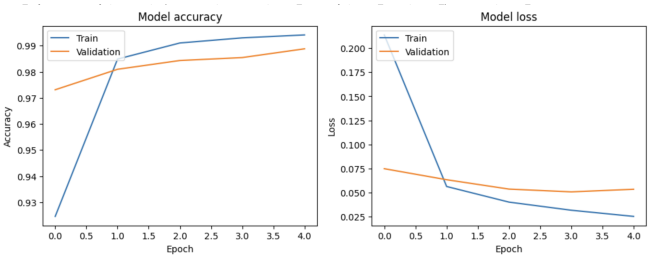

Fig. 4: Plot displaying the accuracy and loss of the fine-tuned LSTM model over each epoch.

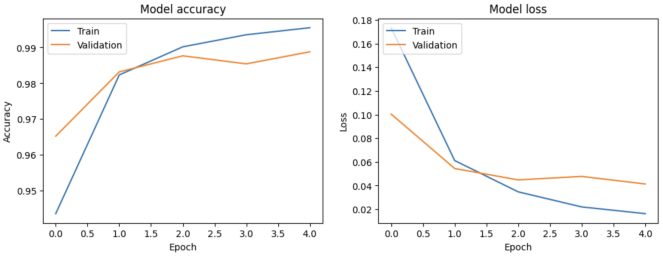

Fig. 5: Plot displaying the model accuracy and loss of the fine-tuned simple RNN model over epochs.

emb_dim	rnn_units	rnn_type	optimizer	activation	dropout	accuracy	loss	precision	recall
50	32	LSTM	rmsprop	relu	0.2	0.982063	0.068654	0.943396	0.931677
50	64	LSTM	rmsprop	relu	0.2	0.982960	0.080875	0.949367	0.931677
50	64	LSTM	rmsprop	relu	0.4	0.982063	0.078154	0.960784	0.913043
100	32	LSTM	rmsprop	tanh	0.4	0.980269	0.089143	0.954248	0.906832
100	64	LSTM	rmsprop	relu	0.2	0.980269	0.080028	0.948387	0.913043
100	128	LSTM	rmsprop	tanh	0.2	0.981166	0.094321	0.960526	0.906832
200	32	LSTM	rmsprop	relu	0.4	0.980269	0.096841	0.960265	0.900621

TABLE VI: Table showing the hyperparameters and performance metrics for LSTM model architecture

emb_dim	rnn_units	rnn_type	optimizer	activation	dropout	accuracy	loss	precision	recall
50	32	GRU	rmsprop	relu	0.2	0.989238	0.054347	1.0	0.922078
50	128	GRU	rmsprop	relu	0.4	0.991031	0.048095	1.0	0.935065
50	128	GRU	rmsprop	tanh	0.2	0.990135	0.051226	0.993103	0.935065
100	32	GRU	rmsprop	relu	0.2	0.989238	0.048813	0.993056	0.928571
100	128	GRU	rmsprop	tanh	0.4	0.990135	0.058549	0.993103	0.935065
200	32	GRU	rmsprop	relu	0.4	0.991031	0.049673	0.993151	0.941558

TABLE VII: Table showing the hyperparameters and performance metrics for GRU models

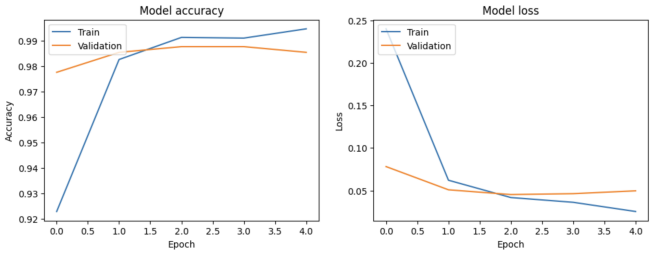

Fig. 6: Plot illustrating the accuracy and loss of the fine-tuned GRU RNN model over epochs.ed