Next Character Prediction using RNN, LSTM, and GRU

Homework 3 - Report

Daniel Jolin
Student ID: 801282735
Intro to Deep Learning
2-28-25
`Github.com/RocketDan11/ml/dl/homework/homework3.ipynb`

Abstract

Next character prediction, a core NLP task, involves predicting the next character given a sequence. This report compares RNN, LSTM, and GRU architectures trained under identical conditions on a character-level dataset. Models were evaluated on sequence lengths of 50 and 20 characters. For $N = 50$, LSTM and GRU outperformed RNN slightly, achieving 87% accuracy. At $N = 20$, accuracy dropped to 58%, highlighting the importance of context length. LSTM and GRU, despite higher complexity, handled sequences better than vanilla RNN. We discuss trade-offs in model complexity, training efficiency, and predictive accuracy, concluding with suggestions for improvement.

0.1 Introduction

Next character prediction is fundamental in NLP, underpinning applications like text generation and autocomplete. Recurrent models such as RNNs, LSTMs, and GRUs are well-suited for sequence tasks. However, RNNs struggle with long-term dependencies due to vanishing gradients, whereas LSTMs and GRUs introduce gating mechanisms to improve memory retention. This report evaluates these models in predicting characters from a text corpus while analyzing the impact of sequence length on performance.

0.2 Methodology

0.2.1 Dataset and Preprocessing

This study utilizes a character-level text dataset for training and evaluating recurrent neural network models (LSTM and GRU) on a next-character prediction task. The dataset consists of a text file (input-text.txt) containing a passage about natural language processing and character prediction. Additionally, a separate Shakespeare dataset is accessed through the shakespear_loader.py script, which downloads text from Shakespeare's works via the Tiny Shakespeare dataset.

Data Representation: Input sequences are one-hot encoded, transforming each character into a binary vector where only the position corresponding to the character's index is set to 1 Target sequences are represented as index tensors

Batch Processing: Data is organized into batches of 32 sequences using PyTorch's DataLoader, with shuffling applied to the training set to improve model generalization.

0.2.2 Models

Three models were implemented using PyTorch:

- **RNN**: A basic recurrent model with 128 hidden units and 2 layers.
- **LSTM**: Includes memory cells with input, output, and forget gates to address long-term dependencies.
- **GRU**: A more compact variant of LSTM with update and reset gates.

0.2.3 Training Configuration

All models were trained using cross-entropy loss and the Adam optimizer (learning rate = 0.001). Training configurations included:

- *sequence_length* = 20: 100 epochs, batch size 32.
- *sequence_length* = 30: 100 epochs, batch size 32.
- *sequence_length* = 50: 100 epochs, batch size 32.

Training was performed on an NVIDIA RTX 3080ti GPU.

0.3 Results and Analysis

0.3.1 Performance for Sample Text

All models achieved high accuracy (∼87%). RNN performed well but had higher validation loss than LSTM and GRU. Training time was similar across models, with GRU slightly faster. Model sizes: RNN (0.24 MB), GRU (0.66 MB), LSTM (0.87 MB).

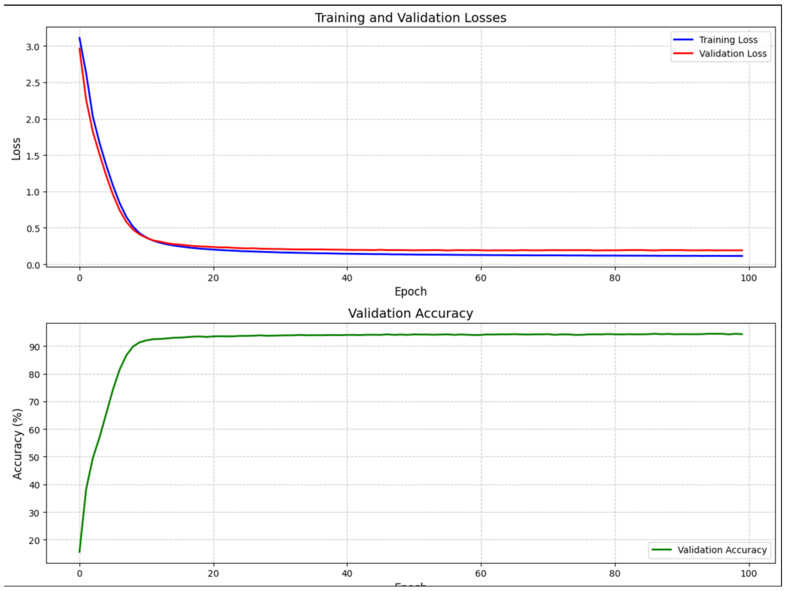
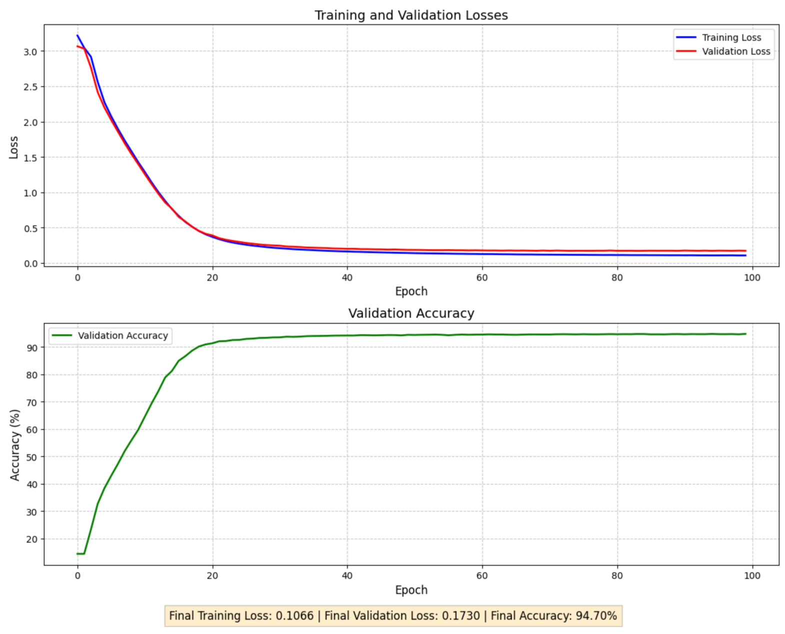
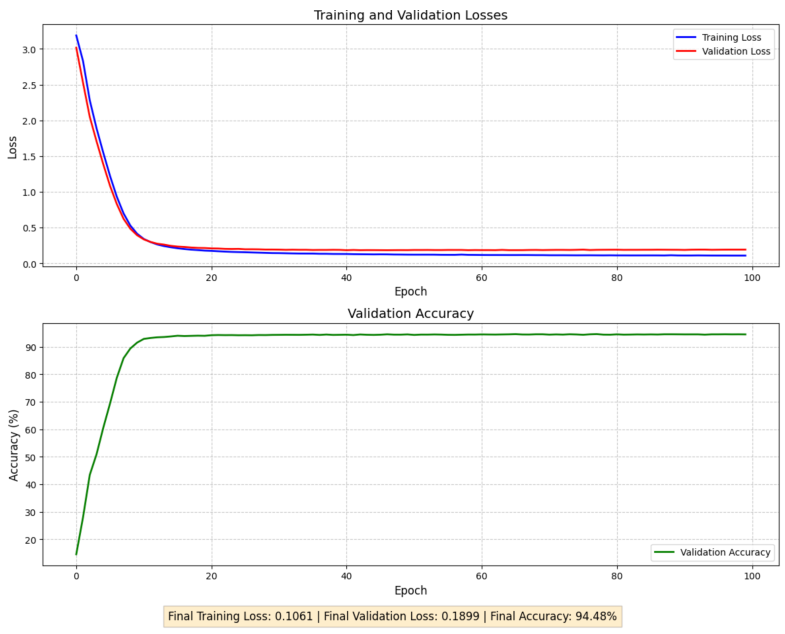


Figure 1: Simple RNN Training Metrics



Final Training Loss: 0.1066 | Final Validation Loss: 0.1730 | Final Accuracy: 94.70%

Figure 2: LSTM Training Metrics

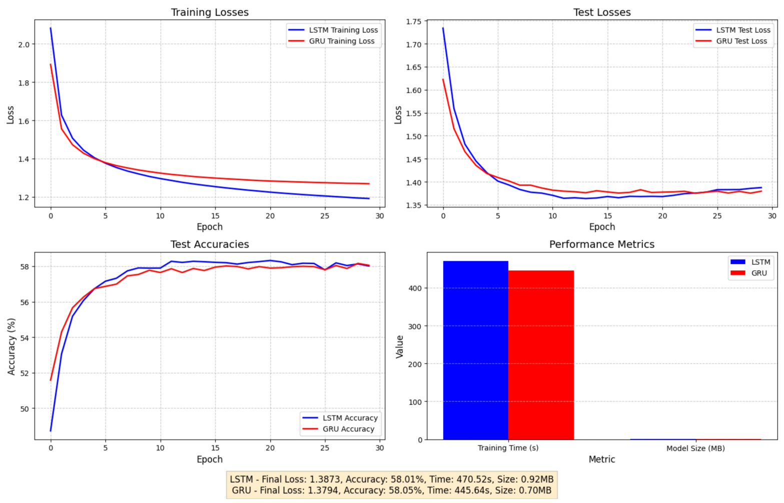


Final Training Loss: 0.1061 | Final Validation Loss: 0.1899 | Final Accuracy: 94.48%

Figure 3: GRU Training Metrics

0.3.2 Performance for Tiny Shakespeare

Accuracy dropped to 58% for LSTM and GRU, showing the impact of limited context and complex training data. GRU trained 10% faster than LSTM while achieving similar accuracy. Shorter sequences resulted in longer training times due to an increased number of training samples.



LSTM - Final Loss: 1.3873, Accuracy: 58.01%, Time: 470.52s, Size: 0.92MB
GRU - Final Loss: 1.3794, Accuracy: 58.05%, Time: 445.64s, Size: 0.70MB

Figure 4: Tiny Shakespeare Comparison

0.4 Discussion

0.4.1 RNN vs LSTM vs GRU

For moderate sequences ($N = 50$), RNN performed well, but LSTM/GRU had lower loss. For shorter sequences ($N = 20$), performance degraded significantly. GRU offers a balance of efficiency and performance, while LSTM is preferable for tasks requiring long-term memory.

0.4.2 Impact of Context Length

Longer sequences provided better predictive accuracy, confirming that more context benefits sequence models. Shorter sequences introduced significant uncertainty.

0.4.3 Trade-offs and Model Selection

- **RNN**: Fast, simple, but struggles with long dependencies.
- **LSTM**: Handles long dependencies well but requires more computation.
- **GRU**: Similar to LSTM but computationally more efficient.

Choice depends on the task: RNN for simple problems, LSTM for complex dependencies, and GRU for efficiency.

0.5 Conclusion

This study compared RNN, LSTM, and GRU on next-character prediction. LSTM and GRU showed superior performance over RNN in handling longer-term dependencies, though all models performed well for moderate sequence lengths. Performance degraded significantly with shorter contexts. GRU proved to be an efficient alternative to LSTM. Future work could explore attention mechanisms or larger datasets to further evaluate these models.