Character-Level Text Generation on Shakespeare and Dickens Using RNN and LSTM Models

Diego Cordova

Department of Cognitive Science · University of California, San Diego

COGS 181: Neural Networks and Deep Learning · Instructor: Zhuowen Tu · Framework: PyTorch

Abstract

This project studies character-level language modeling on two literary corpora, Shakespeare and Charles Dickens, using recurrent neural networks implemented in PyTorch. The goal was to compare a vanilla recurrent neural network (RNN) with a long short-term memory network (LSTM) under a controlled experimental setting. Both models were trained with the same sequence length, hidden size, optimizer, and training budget so that architecture would be the primary variable. Performance was evaluated using held-out test loss, bits-per-character (BPC), and qualitative text generation at multiple sampling temperatures. The results show that the LSTM consistently outperformed the vanilla RNN on both datasets. The best overall model was the Dickens LSTM, which achieved the lowest test loss and the strongest generated samples. These findings suggest that LSTMs are better suited than vanilla RNNs for capturing longer-range structure in literary text and that corpus style also affects modeling difficulty.

1 Introduction

Language modeling is the task of predicting the next token in a sequence given the preceding context. While many modern systems operate at the word, subword, or token level, character-level modeling remains a useful setting for understanding sequence learning at a finer resolution. In a character-level model, the network must learn spelling, punctuation, whitespace, capitalization, formatting, and longer-range sequential structure directly from raw text. This makes character-level generation a compact but meaningful benchmark for studying recurrent neural networks.

Character-level modeling is especially appropriate for a course project because the effects of model quality are easy to observe. Weak models often produce broken words, unstable punctuation, or repetitive fragments. Stronger models begin to generate word-like structure, recognizable sentence flow, and stylistic patterns that resemble the training corpus. Literary corpora are particularly useful in this setting because they have distinctive rhythms and formatting conventions. Shakespeare contains dramatic speaker labels, short dialogue turns, and abrupt scene-like transitions, whereas Dickens tends to follow a more narrative sentence structure. Comparing these two corpora makes it possible to study both model behavior and dataset effects.

The primary architectural question in this project is the difference between a vanilla RNN and an LSTM. A standard RNN updates a hidden state recurrently and can, in theory, preserve information over time. In practice, vanilla RNNs often struggle with long-range dependencies because information can decay as sequences become longer. LSTMs address this limitation through gated memory mechanisms that regulate what information is stored, updated, and forgotten. As a result, LSTMs are widely expected to perform better on sequence modeling tasks that require longer memory.

This project investigates two questions. First, does an LSTM outperform a vanilla RNN for character-level literary text generation when both are trained under the same conditions? Second, does the dataset itself influence how easily the model learns character-level structure? To answer these questions, four controlled experiments were performed: RNN and LSTM on Shakespeare, and RNN and LSTM on Dickens. The models were evaluated using test loss, BPC, loss curves, and generated samples at different temperatures.

2 Method

The project was implemented in PyTorch and extended the recurrent sequence-modeling ideas used in the course homework. Two text datasets were used. The first was Shakespeare, stored as a plain-text file from the course materials. The second was Charles Dickens's *Great Expectations*, obtained as a Project Gutenberg plain-text file and cleaned to remove the Gutenberg header and footer. Using these corpora allowed the project to compare not only architectures but also two different literary styles: dramatic dialogue and narrative prose.

For each corpus, preprocessing was performed at the character level. The vocabulary was built directly from the unique characters in the dataset rather than from a fixed preselected character set. Each character was assigned an integer index, and input sequences were represented as one-hot tensors. The learning task was next-character prediction: given a sequence of 128 characters, the model predicted the next character at every time step. Targets were represented as integer indices and training used cross-entropy loss.

Each corpus was divided into training, validation, and test sets using an 80/10/10 split. During training and evaluation, random subsequences were sampled from the appropriate split. This allowed the project to measure generalization on unseen text rather than reporting only training performance. Test loss was converted to bits-per-character by dividing cross-entropy loss by the natural logarithm of 2. BPC is a standard metric for character-level modeling because it provides an interpretable measure of how efficiently the model predicts the next character.

Two recurrent architectures were compared. The first was a vanilla RNN implemented with `nn.RNNCell`, followed by a linear layer that mapped the hidden state to logits over the character vocabulary. The second was an LSTM implemented with `nn.LSTMCell`, also followed by a linear output layer. For the four main experiments, both architectures used hidden size 128 so that the comparison remained controlled. All runs used Adam with learning rate 0.005, sequence length 128, and 3,000 training iterations. Training loss was recorded every 100 iterations, and validation loss was measured every 500 iterations. The notebook also saved generated text, model summaries, and loss plots for later analysis.

In addition to quantitative evaluation, qualitative text generation was performed at temperatures 0.5, 1.0, and 1.2. Lower temperature concentrates probability on high-likelihood characters and generally produces more stable but more repetitive text. Higher temperature increases diversity but also increases noise. This qualitative analysis was important because text generation quality cannot be fully summarized by a single scalar metric.

3 Experiments

The core experimental design consisted of four runs with all hyperparameters fixed except for dataset and recurrent architecture. The shared settings were sequence length 128, hidden size 128, learning rate 0.005, and 3,000 training iterations. The resulting four-run comparison is shown in Table 1.

Run Name	Model	H	Iters	Test Loss	BPC
shakespeare_rnn_h128	RNN	128	3000	2.1804	3.1456
shakespeare_lstm_h128	LSTM	128	3000	1.9429	2.8030
dickens_rnn_h128	RNN	128	3000	2.0080	2.8970
dickens_lstm_h128	LSTM	128	3000	1.7034	2.4575

Table 1: Test performance across the four core runs. H = hidden size.

Figure 1. Final performance comparison across the four core runs

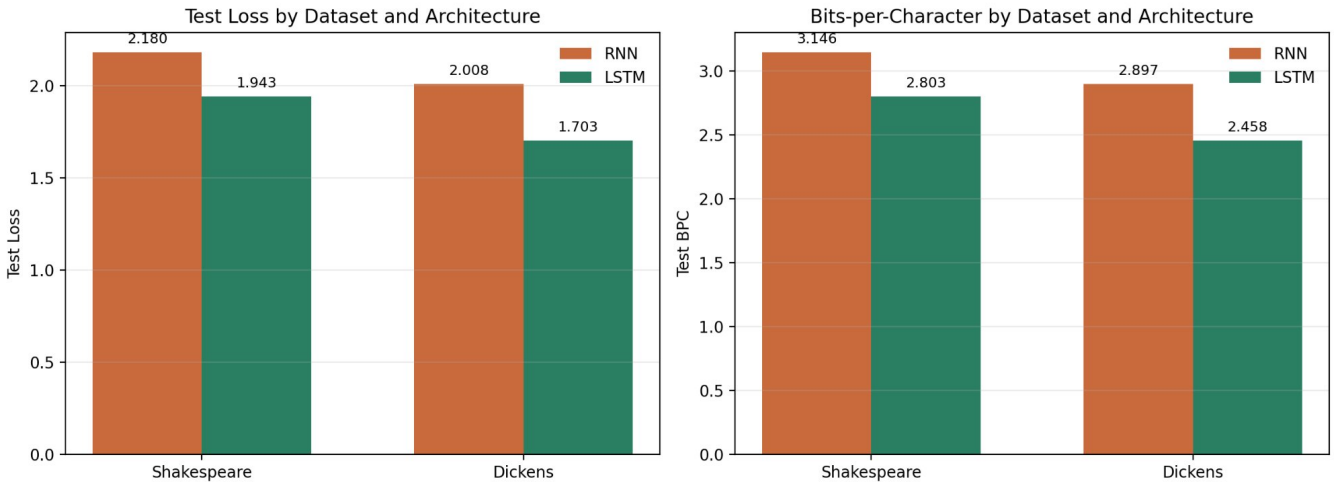

Figure 1: Final test performance across the four core experiments. For both Shakespeare and Dickens, the LSTM achieved lower test loss and lower bits-per-character than the vanilla RNN, with the Dickens LSTM performing best overall.

The first major result is that the LSTM outperformed the vanilla RNN on both datasets. On Shakespeare, the RNN achieved test loss 2.1804, whereas the LSTM reduced this to 1.9429, an improvement of about 10.9%. On Dickens, the RNN achieved test loss 2.0080 and the LSTM reduced it to 1.7034, an improvement of about 15.2%. The BPC values follow the same pattern, with lower values for the LSTM in both cases. These results support the expectation that LSTMs capture sequential context more effectively than vanilla RNNs.

This pattern is also summarized in Figure 1, which provides a compact visual comparison of the final performance across the four runs. The figure makes it immediately clear that the LSTM outperforms the vanilla RNN on both datasets and that Dickens yields lower final error than Shakespeare under both architectures.

The second major result is that Dickens was easier to model than Shakespeare for both architectures. Under the RNN architecture, Dickens achieved lower test loss than Shakespeare by about 7.9%. Under the LSTM architecture, Dickens achieved lower test loss than Shakespeare by about 12.3%. One likely explanation is that the Dickens corpus is more regular at the character level, while Shakespeare includes dramatic formatting, speaker changes, and stylistic shifts that make next-character prediction more difficult.

The recorded loss histories support the same conclusion. For Shakespeare, the final recorded validation loss was about 2.1191 for the RNN and 1.8368 for the LSTM. For Dickens, the final recorded validation loss was about 2.0417 for the RNN and 1.7512 for the LSTM. Thus, the LSTM advantage was visible not only on the final test set but also during validation throughout training.

Figure 2. Training and validation loss curves for RNN vs LSTM

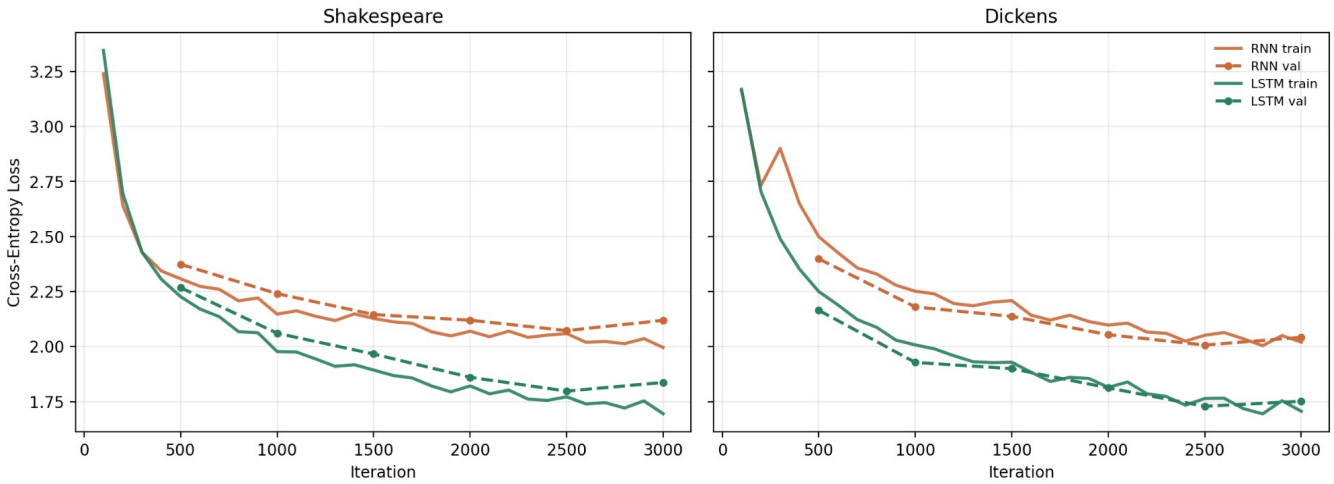

Figure 2: Training and validation loss curves for the Shakespeare and Dickens experiments. In both datasets, the LSTM converged to lower validation loss than the vanilla RNN, supporting the final test-set results.

Figure 2 complements the endpoint comparison by showing the learning dynamics during training. In both corpora, the LSTM maintains a lower validation-loss trajectory than the vanilla RNN, which strengthens the interpretation that the LSTM advantage is stable across training rather than appearing only at the final evaluation step.

Qualitative samples further strengthened the quantitative findings. At temperature 0.5, both models generated more readable output, but the RNN often became repetitive and relied on short recurring fragments. For instance, the Shakespeare RNN frequently repeated function-word patterns such as “and and and,” producing text that looked locally plausible but lacked broader structure. The Shakespeare LSTM, although still imperfect, preserved line breaks and dialogue-style formatting more effectively. The Dickens RNN generated word-like fragments but often drifted into unstable character combinations. In contrast, the Dickens LSTM produced the strongest narrative flow among all four runs, with more plausible punctuation and longer readable stretches.

Sampling temperature also behaved as expected. Temperature 0.5 produced safer and more stable outputs but at the cost of repetition. Temperature 1.0 gave the best balance between coherence and diversity. Temperature 1.2 generated more creative but noisier samples, often with broken words and abrupt transitions. This reinforces the idea that final text quality depends not only on the trained model but also on the sampling strategy used during generation.

4 Conclusion

This project investigated character-level literary text generation using two recurrent architectures, a vanilla RNN and an LSTM, on Shakespeare and Dickens corpora. The results consistently favored the LSTM. Across both datasets, the LSTM achieved lower test loss and lower BPC than the vanilla RNN, and its generated samples showed stronger local coherence, more convincing punctuation, and better preservation of corpus style. The best-performing configuration was the Dickens LSTM, which achieved the strongest overall quantitative and qualitative performance.

The dataset comparison was also informative. Dickens was easier to model than Shakespeare under both architectures, suggesting that literary form and formatting conventions affect character-level prediction difficulty. Shakespeare’s dramatic speaker structure, abrupt transitions, and stylistic variation likely make it a more challenging sequence modeling problem than Dickens’s more narrative style.

There are several limitations to this work. First, the models were relatively small and were trained for a modest number of iterations. Larger hidden sizes, deeper models, or longer training may improve results further. Second, the project used one Dickens work and one Shakespeare corpus rather than a broader collection of texts. Third, evaluation focused on loss, BPC, and qualitative inspection rather than human preference ratings or more advanced statistics.

Future work could extend the project by comparing hidden sizes such as 128 versus 256, adding a GRU baseline, or using additional datasets with different styles, such as poetry, legal writing, or technical prose. Even within the limited scope of this project, the experiments clearly demonstrate that recurrent architecture matters and that LSTM provides a measurable advantage for character-level literary text generation.

References

[1] Hochreiter, S., and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.

[2] Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. *Blog post*. http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[3] PyTorch Documentation. Recurrent neural network cells and tensor operations. https://pytorch.org/docs/

[4] Dickens, C. *Great Expectations*. Project Gutenberg plain-text edition.

[5] Shakespeare corpus distributed in course homework materials.