Enhancing Character-Level Recurrent Neural Networks for Text Generation: A Comparative Analysis on LSTM Architectures and Data Augmentation

Wallace Elis Wayne Wefel WEWEFEL@UCSD.EDU
The Department of Cognitive Science
University of California San Diego
9500 Gilman Dr.
La Jolla, CA 92093, USA

Abstract

In this analysis, I explore the effectiveness of character-level recurrent neural networks (Char RNN) for text generation. The study focuses on improving model performance through alterations in Long Short-Term Memory (LSTM) architectures, including comparing single-layer and double-layer LSTMs, then comparing the superior model with and without data augmentation and regularization techniques. I use the Tiny Shakespeare dataset (Karpathy, 2015) to evaluate the performance of the three separate models through key metrics such as loss, perplexity, and accuracy. The results suggest that the single-layer LSTM model provides significantly greater performance compared to the double-layer LSTM model. Subsequently, I aim to improve the single-layer model through the incorporation of data augmentation and regularization. My findings insist that the additional techniques may have slightly enhanced overall model performance as indicated by lower final loss and perplexity values, as well as a generally less variable training cycle in comparison to the model without data augmentation or regularization. Overall, this paper raises insight into model selection and fine-tuning as I seek to contribute and gain knowledge in the fields of natural language processing, neural computation, and text generation.

Keywords: Character-level RNN, LSTM, Text Generation, Machine Learning, Neural Networks

1 Introduction

The utilization of deep learning in recent years has caused immense breakthroughs in a variety of automated decision-making issues. Recurrent neural networks specifically - characterized by its ability to direct information both forward and backward in its multiple layers of artificial neurons in a neural network - have illustrated great potential for generating text sequences. This study addresses character-level RNNs, which process and generate text one character at a time. Furthermore, I focus on optimizing single-layer LSTM architectures through data augmentation techniques, including character swapping and random deletion, and dropout for regularization. By analyzing the three models on the Tiny Shakespeare dataset, I seek to acquire insight towards optimal techniques and parameters in text generation.

2 Methodology

2.1 Data Preprocessing

I chose the Tiny Shakespeare dataset for this study's analysis as it includes 40,000 lines of text from a variety of Shakespeare plays, amounting to over 1.1 million characters. The preprocessing of the data involved removing special characters and extra whitespaces with Unidecode before being tokenized into individual characters, allowing it to be used as input for the models.

2.2 Model Architectures

Initially, I established two LSTM Char RNN models (a single-layer LSTM and a double-layer LSTM) to discover the optimal number of LSTM layers. All other hyperparameters - learning rate, number of iterations, embedding dimensions, hidden dimensions, and batch size - were kept the same for each model in order to maintain consistency and focus on the effect from layer quantity.

2.3 Data Augmentation and Regularization

After discovering the optimal number of layers, I introduced data augmentation in an effort to increase variability of training data. The data augmentation techniques used consisted of character swapping and random deletion. Additionally, a dropout with rate of 0.5 was implemented into the RNN for regularization. Data augmentation and regularization was only used on the superior single-layer LSTM model.

3 Experiments and Results

3.1 Training Setup

All three LSTM models were trained on 4000 iterations with loss, perplexity, and accuracy recorded at each interval, and model checkpoints were saved at every 200 iterations. Each model utilized the Adam optimizer, a learning rate of 0.005, embedding and hidden dimensions of 128 each, and a batch size of 32.

3.2 Layer Number Comparison

The first step of the study was to optimize one of the most important hyperparameters in our RNN: number of layers. The single-layer LSTM without data augmentation reached a final loss value of 1.615, which scores notably superior to the double-layer LSTM without data augmentation - final loss of 1.789. See Table 1 for more information.

	Single-layer	**Double-layer**
Loss	1.615	1.789
Perplexity	5.030	5.981
Accuracy	0.535	0.475

Table 1. The results illustrate that the single-layer model was more effective for this task as indicated by the three key metrics. Note that these results are before data augmentation was introduced.

3.3 Data Augmentation Effect on Single-layer LSTM

The next objective is to build upon our single-layer model with data augmentation.

	Without	**With**
Loss	1.615	1.605
Perplexity	5.030	4.977
Accuracy	0.535	0.525

Table 2. The results illustrate that the model with data augmentation was approximately the same or slightly more effective for this task as indicated by the three key metrics. Note that the "Without" class refers to the single-layer LSTM without data augmentation and regularization, and the "With" class refers to the single-layer LSTM with data augmentation and regularization.

Although Table 2 shows rather negligible difference after data augmentation, it is critical to visualize and understand the entire process and any patterns in training that final metric values cannot properly express. See Figure 1 for more information on the changes in loss throughout each training iteration.

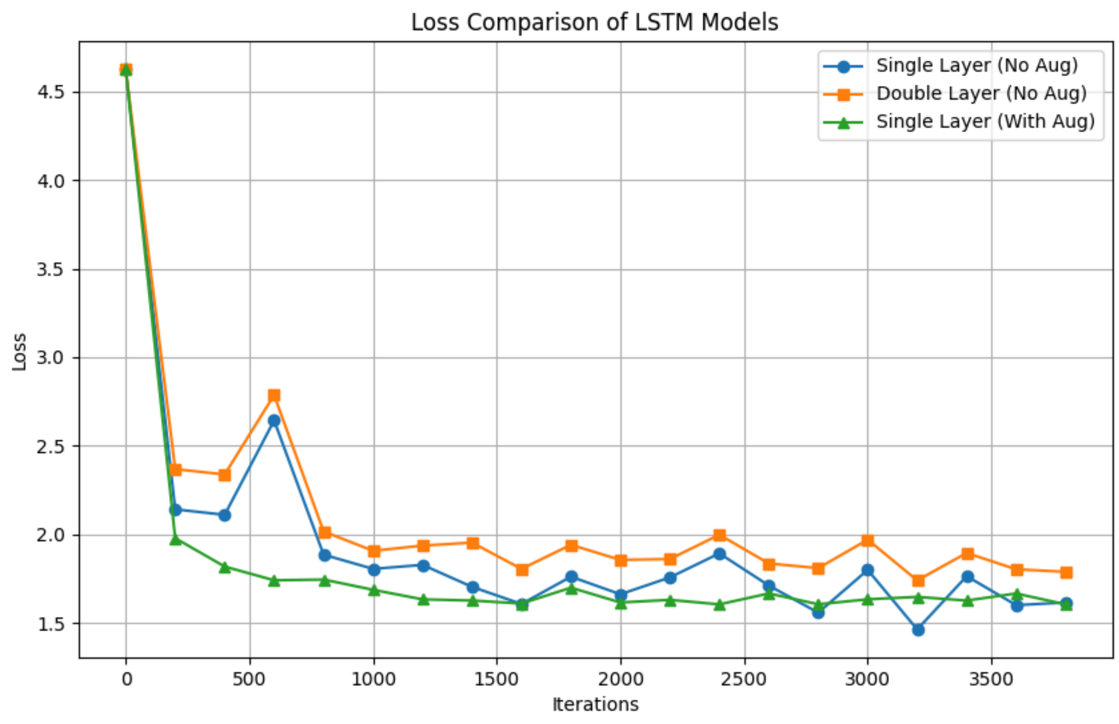

Figure 1. The figure shows all three models, with the double-layer model consistently performing worse and both versions of the single-layer model producing similar results.

Considering Figure 1, we are able to visualize the volatility in the single-layer with no augmentation or regularization. The model with augmentation and regularization is exceptionally more stable in its results throughout the entire training process with significantly more model checkpoints that have a smaller loss value.

4 Discussion

As depicted by both Table 1 and Figure 1, the single-layer LSTM is superior to its double-layer LSTM counterpart in effectiveness in this text generation problem. Moreover, the effect of data augmentation techniques and regularization is still somewhat nuanced. According to Figure 1, data augmentation and regularization appears to have assisted in overall stability in the training process but does not grant an outstandingly greater difference in final performance compared to the same model without augmentation and regularization. It is not clear what the primary cause of the decreased volatility is. Data augmentation increases diversity of the training data by transforming existing data into new samples, but it is uncertain whether it would have this great of an effect in reducing the volatility in loss between iterations. It is possible that the dropout technique I used for regularization helped improve generalization since it randomly "drops out' neurons at each cycle, preventing dependence on specific neurons by the network.

5 Conclusion

This research gives an extensive comparison between single and double-layer LSTM architectures for Char RNN while also demonstrating the significance of data augmentation and regularization techniques. Although results from this analysis illustrate notable differences between each of the three models, it is important to consider further investigation through re-training of each model as well as changes in hyperparameters to validate our findings. Key topics for future fine-tuning include expanding training data, optimizing learning rate, embedding dimensions, hidden dimensions, and comparing single, double, and triple-layer models that each have varying optimized hyperparameters.

Acknowledgements

I would like to make a special note of appreciation to Sebastian Raschka on YouTube for his video titled "L19.2.2 Implementing a Character RNN in PyTorch – Code Example," as well as his other two videos in the Char RNN series. Raschka provided immense help in both my general understanding of the topic and how to get started with my code.

References

Karpathy, A. (2015). Tiny Shakespeare Dataset. Retrieved from https://github.com/karpathy/char-rnn/blob/master/data/tinyshakespeare/input.txt on June, 2024.

Raschka, S. (2021). Character RNN Example. Retrieved from https://github.com/rasbt/stat453-deep-learning-ss21/tree/main/L19/character-rnn on June 15, 2024