Garfield Comic Text Generation

Noah Grant
UCSD
nogrant@ucsd.edu

Abstract

Garfield comics are a well documented, colourful, and varied collection of works which hold a consistent voice through the comics. Spurred on by a repository of the comics in plain text, a recurrent neural network (RNN) using long short-term memory (LSTM) cells was developed, trained off a cleaned version of this database. Characters were mapped with one-hot encoding and sequentially ordered to generate text mimicry of the comics. Attempting to follow more nuanced text development (comic styling of audio gags, character consistency, etc.) was the primary goal, pursued with higher than normal hidden layer and iteration counts. Although the generated outputs mimic the comic style of audio gags, cadence, and general style, the final product still falls short of true comic emulation and further pursuit of LSTM cells in short text development should be considered.

1 Introduction

In light of "recent" advances in neural networks, particularly LSTM networks, have transformed the landscape of natural language processing by enabling the generation of coherent and contextually relevant text. In this project, I explore the use of LSTM-based RNNs for character-level text generation with a specific focus on emulating the succinct and stylistically distinctive language of Garfield comics.

LSTMs, introduced by Hochreiter and Schmidhuber (1997)[1] , are a specialized type of recurrent neural network designed to overcome the limitations of traditional RNNs, most notably the vanishing gradient problem. By incorporating gated mechanisms (forget, input, and output gates), LSTMs are capable of retaining long-term dependencies in sequential data. This mathematical framework allows the network to capture both short-term syntax and long-range contextual dependencies, making it particularly well-suited for tasks such as text, or in this case comic, generation.

In the context of Garfield comics, which are characterized by their brevity and consistent "voice"/ consistency of character, a character-level approach is especially appropriate. Unlike longer textual documents, the compact nature of comic strips requires the model to effectively learn and reproduce subtle stylistic cues and the unique rhythm of dialogue and captions. The use of character-level modeling facilitates the generation of text that mirrors the original comics' concise and impactful style, even when the training sequences are relatively short.

My approach begins with cleaning the Garfield comic plain text archive, stripping out extraneous comic reference markers and converting the text into one-hot encoded sequences. These sequences serve as the input-output pairs for training the LSTM network, where the objective is to predict the subsequent character given a sequence of previous characters. Building on the foundation laid by works such as Karpathy's explorations in char-level RNNs[2], our implementation employs a multi-layer LSTM architecture with dropout regularization to enhance generalization and prevent overfitting.

In experimenting with LSTMs, I aim to explore the flexibility and use cases of LSTM networks in modeling and generating short, high-impact and relevant textual content. The integration of advanced neural architectures with domain-specific training data underscores the potential of neural text generation in applications ranging from creative writing to automated comic strip generation.

2 Method

I developed my text generator in multiple stages: data preprocessing, model architecture design, training procedure, evaluation, and graphical analysis. Each of these stages is implemented in Python using PyTorch, numpy, and matplotlib, and the code is structured to enable both automated training and interactive text generation.

2.1 Data Preprocessing

The input data is derived from a cleaned corpus of Garfield archived plain text comic script. The text file is then loaded post stripping and I compute its length to assess the dataset size. Python's string.printable was used to define the set of all possible characters, ensuring that the model can account for punctuation, digits, and special symbols (see Data Preparation in the code, [4]). A random sequence extraction function (get_random_seq) is employed to generate input sequences of fixed 128 character length. These sequences are subsequently transformed into two representations:

One-Hot Encoding: The function seq_to_onehot converts each character into a one-hot vector, forming a tensor with shape (sequence length, 1, number of characters). This representation is used as input to the network.

Index Encoding: The function seq_to_index maps characters to their corresponding indices for target prediction.

These steps ensure that the sequential structure of the text is preserved and that the data is appropriately formatted for the recurrent network.

2.2 Model Architecture

I implement a multi-layer LSTM network using PyTorch's nn.LSTM module. The network is designed with two LSTM layers and a hidden size of 256, augmented by a dropout layer (dropout rate of 0.2, lower end for quicker computation times given the short nature of comic text) between LSTM layers to mitigate overfitting. The LSTM's output at the final time step is passed through a linear layer to produce a probability distribution over the character vocabulary. This architecture is inspired by prior work on character-level RNNs [2] and the seminal LSTM design [3]. The relevant code can be found in the Net class. Of particular relevance to understanding and applying LSTM architecture, the following two equations come to mind:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \tag{1}$$

Equation (1) demonstrates cell state update and retention using a forget gate *ft* $\odot$ *Ct* and update/ input gate *it* $\odot$ *Ct*. The sigmoid function forget gate finds values between 0 and 1 to "remember" what is decided by previous calls, and the update/ input gate then uses a tanh base function to retain specific values within it's range.

$$h_t = o_t \odot \tanh(C_t). \tag{2}$$

Equation (2) is the hidden state equation, which revolves around output gate *ot* and activation check *tanh(Ct)*. The output gate measures what is exposed from the hidden state (in bounds of sigmoid) and the hyperbolic tangent function based activation check checks for remaining values between -1 and 1 to retain and express.

2.3 Training Procedure

Training is conducted iteratively 20,000 times. In each iteration, a random sequence is extracted from the text, and the network is trained to predict the next character at each time step using the cross-entropy loss function. The train_step function encapsulates the forward pass through the network, the loss calculation, and the backpropagation step using the Adam optimizer. To monitor progress, we accumulate the average loss over fixed intervals (every 5,000 iterations) and record the time taken for each interval. This is then provided with a sample of generated text to ensure the model is still working properly and reducing error. The output provides insight into both the convergence behavior of the loss function and the training efficiency.

2.4 Evaluation and Text Generation

An evaluation function (eval_step) is implemented to generate new text sequences on command. Given an initial prompt, the function builds up the hidden state by processing the prompt through the network. It then generates subsequent characters by sampling from the softmax probability distribution over the vocabulary, adjusting randomness via a temperature parameter. This iterative process yields a generated text sequence that aims to mimic the style and structure of the Garfield comics. This creates the output text that is seen alongside the function results from the previous procedure.

2.5 Graphical Analysis

To visualize the training process, I record both the average loss and the time duration for every logging interval. Using matplotlib, I plot:

Loss over Iterations: This graph illustrates how the training loss decreases over time, serving as a measure of model convergence.

Training Duration per Interval: This plot shows the time taken for each interval, providing an indication of training efficiency and computational performance.

The corresponding code sections for plotting are integrated into the main training script, and displayed for reader understanding later.

3 Experiment

In this section, I describe the experimental setup, present the evaluation metrics, and discuss the results obtained from training the character-level LSTM model on the Garfield comics corpus. Ground will be retread such that users can read just this section and still recreate the same results.

3.1 Experimental Setup

The model was trained on a processed corpus of Garfield comic text. The text was first cleaned to remove extraneous comic reference data, and then converted into sequences of characters using one-hot encoding and index representations. The training procedure consisted of 20,000 iterations, with performance monitored every 5,000 iterations. During each interval, the average cross-entropy loss and the duration of the training interval was recorded. The model architecture comprises a two-layer LSTM with a hidden size of 256 and dropout regularization (dropout rate = 0.2). The Adam optimizer with a learning rate of 0.003 was employed during training, mildly higher than usual such that the large amount of iterations could be developed quickly and such that the training accuracy reflects the brevity of text generation desired [5].

3.2 Evaluation Metrics

The model was assessed using the following metrics:

- **Training Loss:** The computed average cross-entropy loss over each sequence during training. A steady decrease in this loss indicates that the model is effectively learning the character distributions in the text.
- **Training Duration:** The measured time required to complete each training interval. This metric serves as an indicator of the model's computational efficiency and consistency in training.

3.3 Results

Figure 1 shows the decrease in the average training loss over the 20,000 iterations. The loss curve demonstrates a clear trend of convergence, confirming that the model gradually learns to predict subsequent characters more accurately.

Figure 2 illustrates the time taken per 5,000-iteration interval. The recorded durations are consistent, indicating stable computational performance throughout the training process. The times, however, are quite high.

Figure 3 provides a sample output generated by the model after training. The output, though not perfect, almost reflects the succinct and punchy style typical of Garfield comics. Further tuning of the temperature parameter and extended training may help in producing more natural and "Garfield" like language.

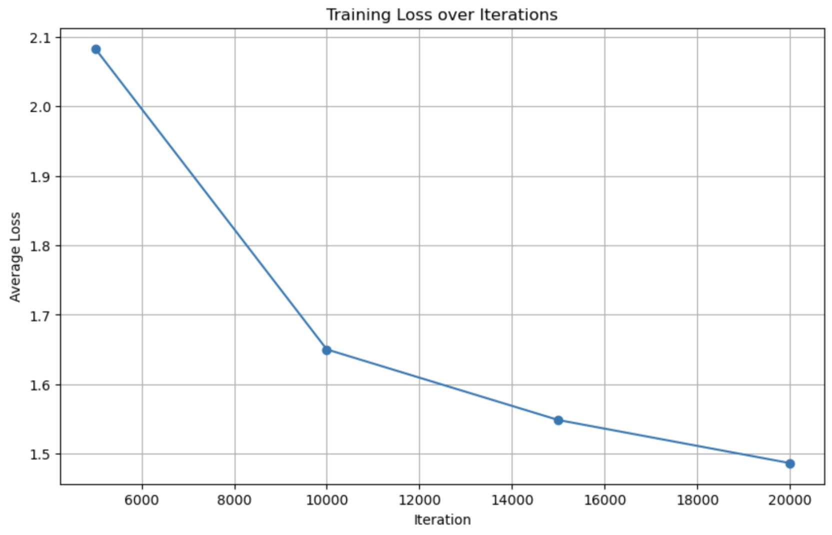


Figure 1: Average training loss over iterations.

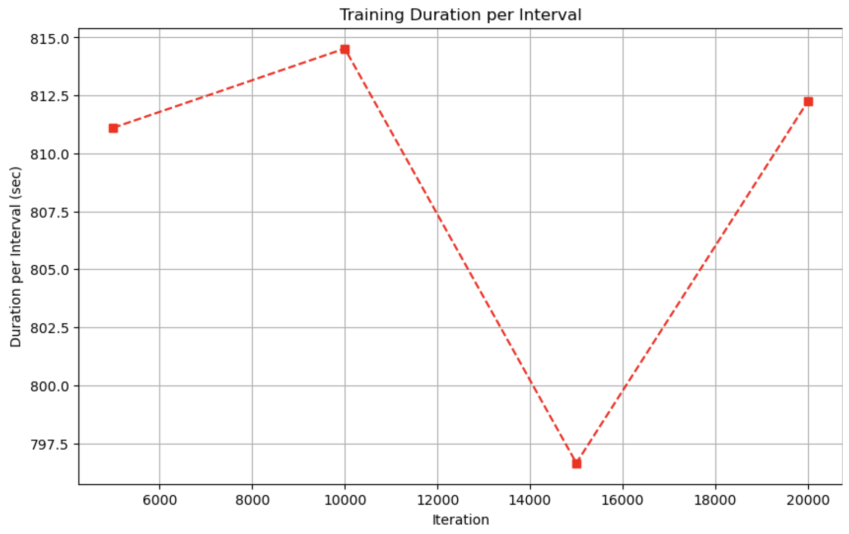
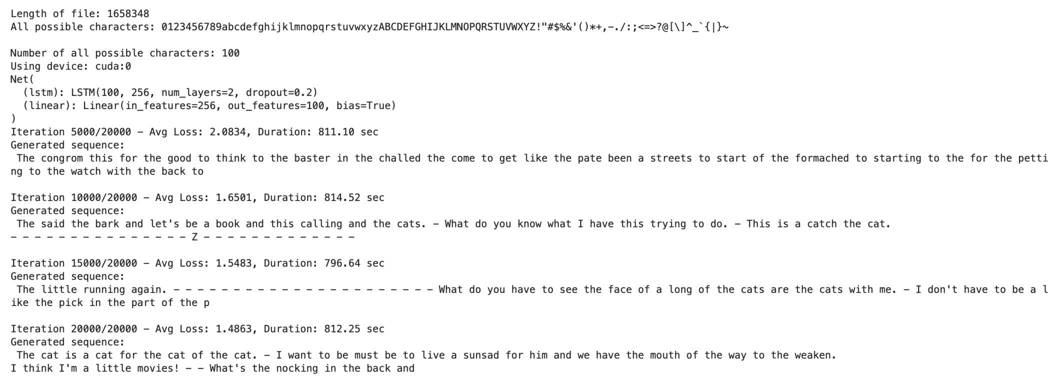


Figure 2: Training duration per 5,000 iterations.



```
Length of file: 1658348
All possible characters: 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

Number of all possible characters: 100
Using device: cuda:0
Net(
  (lstm): LSTM(100, 256, num_layers=2, dropout=0.2)
  (linear): Linear(in_features=256, out_features=100, bias=True)
)
Iteration 5000/20000 - Avg Loss: 2.0834, Duration: 811.10 sec
Generated sequence:
 The congrom this for the good to think to the baster in the challed the come to get like the pate been a streets to start of the formached to starting to the for the petti
ng to the watch with the back to

Iteration 10000/20000 - Avg Loss: 1.6501, Duration: 814.52 sec
Generated sequence:
 The said the bark and let's be a book and this calling and the cats. - What do you know what I have this trying to do. - This is a catch the cat.
- - - - - - - - - - - - - - - Z - - - - - - - - - - - - -

Iteration 15000/20000 - Avg Loss: 1.5483, Duration: 796.64 sec
Generated sequence:
 The little running again. - - - - - - - - - - - - - - - - - - - - - - What do you have to see the face of a long of the cats are the cats with me. - I don't have to be a l
ike the pick in the part of the p

Iteration 20000/20000 - Avg Loss: 1.4863, Duration: 812.25 sec
Generated sequence:
 The cat is a cat for the cat of the cat. - I want to be must be to live a sunsad for him and we have the mouth of the way to the weaken.
I think I'm a little movies! - - What's the nocking in the back and
```

Figure 3: Sample output of generated text after training.

3.4 Discussion

The experimental results demonstrate that my LSTM-based text generation system is capable of learning the distinctive style of Garfield comics. The gradual reduction in training loss indicates that the model effectively captures the underlying character-level distributions. Moreover, the stable training durations across intervals (+/- 6sec) suggest that the approach is computationally efficient. Although the generated output captures some stylistic elements of the source material, additional refinements—such as further hyperparameter tuning, deeper architectures, or exploring word-level models—may be necessary to produce text that is more natural and grammatically coherent. The outputs, although at times funny, still require further epochs or iterations to be in line with the nature of Garfield's character, and there may be further ways to hyper-optimize this algorithm such that it doesn't take an hour to finish training.

4 Conclusion

In attempting to develop a character-level text generation system based on LSTM networks, tailored to emulate the succinct and distinctive style of Garfield comics, I feel like I've only found the boundaries of rough generative text approaches to capturing nuanced character interactions. My approach involved extensive data preprocessing, including one-hot encoding of characters and the construction of sequential input-output pairs, followed by training a multi-layer LSTM with dropout regularization. The training process, monitored over 20,000 iterations, demonstrated a gradual decrease in cross-entropy loss, indicating effective learning of character distributions. Graphical analyses further confirmed both the convergence of the model and its computational efficiency.

Despite the promising results, the generated text exhibits occasional inconsistencies and lacks full grammatical coherence. This suggests that while the model captures many stylistic nuances of the original Garfield comics, further refinements—such as exploring deeper network architectures, tuning hyperparameters, or transitioning to a word-level model—could improve the naturalness of the generated language.

Overall, the findings underscore the potential of LSTM-based RNNs for creative text generation, particularly for applications requiring the synthesis of short, high-impact content.

Further work Future work will focus on enhancing the model's linguistic capabilities and exploring alternative architectures to better balance creativity with coherence, along with quicker training times. More effort needs to be spent on recreating the brevity of comic humour and the ability to understand which character is speaking without needing character tags (more levels?).

5 References

[1] `https://www.researchgate.net/publication/13853244_Long_Short-Term_Memory`

[2] `https://karpathy.github.io/2015/05/21/rnn-effectiveness/`

[3] `https://pytorch.org/docs/stable/index.html`

[4] HW5 on RNNs. Huge shoutout to this piece for laying the framework for me to develop off of.

[5] `https://arxiv.org/abs/1412.6980`