Colin Kavanagh

Zhuowen Tu

COGS 185

6/15/24

Rocking Neural Network: Character RNN for Lyric Generation

Abstract:

In this paper, we explore the feasibility of Character Recurrent Neural Networks for generating lyrics for rock songs. The experiments explore multiple layers, dropout methods, learning rates, iterations, and nonlinearity. In the experimentation, a four-layer RNN with dropout layers performed the best of all the models regarding the average loss and time required to train the model.

Introduction:

Neural networks are a unique form of artificial intelligence algorithms that use multiple layers of perceptrons to make calculations and predictions. The world has recently seen "deep" neural networks rise due to their potential and feasibility for various computational problems. Recurrent Neural Networks are a particular type of neural network that considers contextual and sequential input. A typical feed-forward neural network takes in all inputs at once, which isn't suitable for problems that involve particular inputs. Recurrent Neural Networks consider the current input relative to the previous inputs. For example, a regular neural network wouldn't be good for a problem translating a sentence to another language because there isn't a way for the network to consider the order of the words. However, RNNs are built to take sequential input and are great for similar problems. Specifically, Character Recurrent Neural Networks use characters as sequential input and try to predict the next character in a sequence. Compared to words, there

are fewer possibilities for characters (about 100 instead of tens of thousands of words); however, there are more calculations for each sentence. In this project, I use a Character Recurrent Neural Network to generate lyrics from rock songs.

Methods:

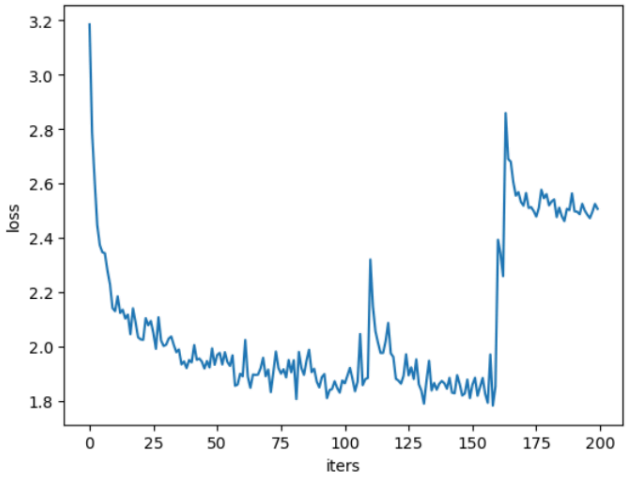
For this project, I used the Multi-Lingual Lyrics for Genre Classification dataset by Matei Bejan. This dataset contains over 290k different lyrical samples from various genres and languages. I needed to preprocess this data to use it for our Char-RNN. First, I drop all of the columns instead of "Genre," "Language," and "Lyrics." Second, I filtered the entries only to include lyrical entries in English and considered "rock" songs. After applying these filters, the data contains over 107k song samples for rock lyrics. I then removed all non-ASCII characters from the lyrics. Finally, I exported the song lyrics to a text file named "trainlyrics.txt."

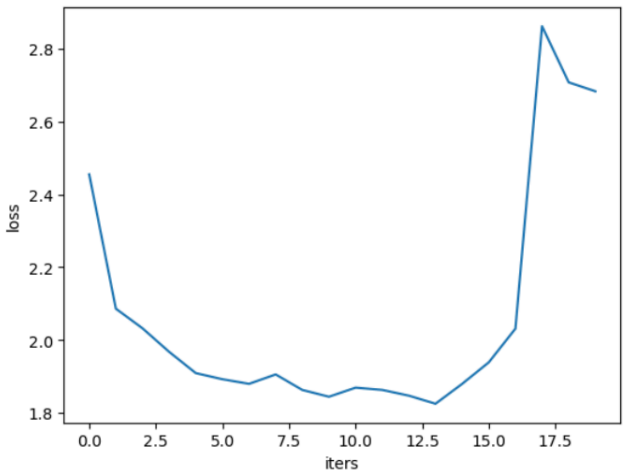
I used the skeleton code from the COGS 185 Homework 3 Bonus Problem 1 to create the network. This code gets random sequences of excerpts from the song lyrics to train the network. Then, one hot encodes the characters in the sequence for input into the network. Then, the network trains on the excerpt, learning the various probabilities for the subsequent sequences of characters. Throughout all of these iterations, it calculates the Cross-Entropy Loss of results and uses it to teach the network further the correct probabilities. Finally, the network makes a final prediction to give a general idea of its performance on the training data.

As mentioned before, the primary type of network I will use is an RNN, as implemented in PyTorch. I will also experiment with some of the various hyperparameters, such as dropout rates, learning rates, layers, and iterations. However, I will also experiment with using Gated Recurrent Units (GRU). Since many of these networks took several hours to train, I made the networks gradually more advanced, making multiple changes simultaneously.

Results:

The experiments begin by using the baseline RNN provided to us in the skeleton code. This baseline network is a one-layer RNN, followed by a linear layer. This network used an Adam optimizer with a learning rate 0.005 and a Cross Entropy Loss function with 20k iterations. While this is a simple model, it is a good baseline for determining the difficulty of the problem and the complexity of the model required to solve it. This network provided some of the most interesting results out of all the networks experimented with. These results are because the overall loss spiked multiple times throughout training, rendering previous progress null and void. Below are the graphs of the two trials I did with this model and a generated text sample. A massive spike around the 16k iteration mark starts training from the beginning. Because of this, the final sample generated is mainly gibberish, only determining typical word length and spacing at best. I thought this could be a coincidence; maybe the model was training on a bad batch of data that I couldn't filter out. So, I reran the experiment to see if I could get different results. However, I got the same results as before.





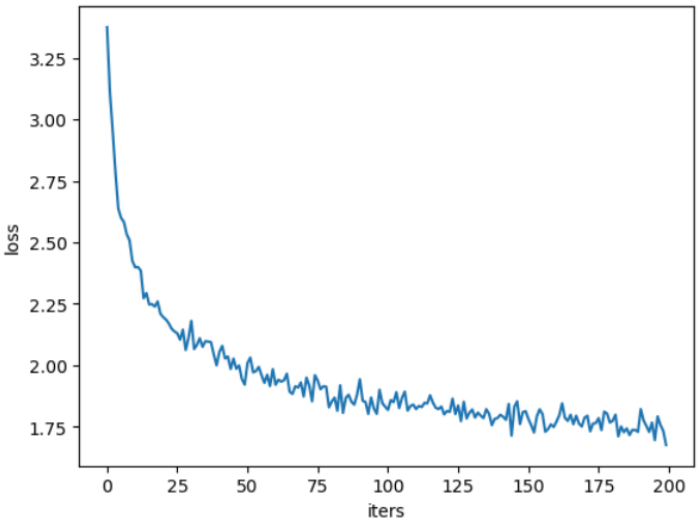

```
Warwacghau dess y,reno mn
T Sou the dohe
Sthn'vogts
Anon, beatgour ykie thswt
Idecay,e konasthed mf th ap oum tom absh  yanrynse on'mo
Irt
(oustsert'sher k k sowe youl
Sangs'sae't ce thkn ilnuy f'dn ca eon e utog'tth
s'trunn cot daat stir Icy as be b Wfrire my urnt Yoe yosey thuts'th
(Mein
I laoc aown aayyou

aymy fastl
Cot tou'tan
Sath
o nwatnk
Jabet ydo
tsk
( thno ou,
```

(Graphs of loss for each iteration, plus final text sample)

This sudden spike in loss is caused by the Exploding Gradient Problem. The Problem occurs when the gradients used in backpropagation become extremely large. Because backpropagation involves multiplying the gradients, if a gradient is greater than 1, it can cause a chain reaction to cause the other gradients to “explode.” The Exploding Gradient Problem is potentially why we see such multiple high spikes in loss while training the model. For the next experiment, I will implement ways to prevent the Exploding Gradient Problem.

In the subsequent trial, I change a couple of the hyperparameters to improve on the

previous model. For this model, I decided to make it more complex by using four different RNN layers instead of just one. I added more layers because I didn't think the model was complex enough to understand the nuances of some of the song lyrics with just one RNN layer. Using four layers instead of just one was a good starting point for making the model more complex. I also added dropout layers in between the RNN layers. I added the dropout layers in direct response to the Exploding Gradient Problem we saw in the last trial. The dropout layers allow random inputs into the dropout layers to be randomly "dropped." While removing information sounds counterproductive to our purposes, dropout layers prevent issues like overfitting and exploding gradients because they prevent the network from becoming overreliant on specific inputs. The nonlinearity functions in the model were also changed from the default "tanh" to "ReLU" to prevent the "Vanishing Gradient Problem" from happening, which is the opposite of the Exploding Gradient Problem. Finally, I reduced the learning rate to 0.001 from 0.005. While this is a minor change, I believe it could potentially help with finding the optimal solution if the previous model was overshooting it. Below is a results graph and the final generated text sample.



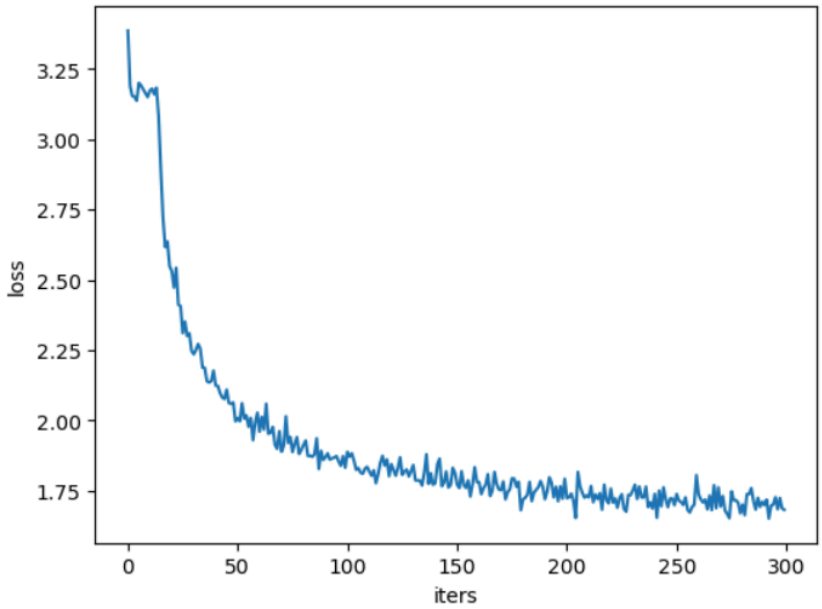

```
Wed all be
Where end is last hroll a had be
Stide shosercount I want
Don't don't fell out I'm sees or to brow no far
it's not me taking alout you grincion one
We've graws

Come, mrough sore ageirs plob wife?
Well it'll deep
who'll wait got a widnarder
I'm so forget to deece a teen feeling
Are me more, the curtter Uy

Wey!, thitk
And goin' my fany ow live ricthing it so a we sloar you
And a cord into to ness it knows pain
Thet sometit must, bling to so turn the munnce's days, but skewing
Oh..'a agoun
Mathing droin as see now, would vea
So eyes thats my bright apay...aw
Ur, you've donna
If I rati
```

(Loss over 20k iterations, final text sample)

As shown by the graph, the Exploding Gradient Problem has wholly vanished. This elimination is probably thanks to the dropout layers added to the model. The model also avoided vanishing gradients, most likely due to adding the ReLU functions. The model's loss also goes gradually downward due to later iterations. While the slope of the curve is low, it doesn't completely flatten out, which implies that better model performance is possible given a more significant number of iterations.

The image shows that the model achieved decent cohesion in rock lyrics. The model understood the lyrics well enough to produce words that an artist would use in rock lyrics. While much of what the model produced would be interpreted as gibberish by English speakers, specific patterns of language and words that the model picks up on could fool a non-English speaker into thinking it was coherent English text. The model's text is similar to "Simlish," a fictional language from the video game series *The Sims*.

Besides the more successful networks, I tried different hyperparameters that created models that were not up to par with some of the other models. In one model, I tried switching the layers from a regular RNN to Gated Recurrent Unit (GRU) layers, which use specialized "gates" to prevent overfitting and the vanishing gradient problem. I opted to use GRU instead of LSTM layers because they are easier and faster to train but usually perform about the same. As the graph below indicates, the GRU network resulted in a decent performance but a flatter loss curve near the end of training. These results imply less model potential for performance, given a more significant number of iterations than the regular RNN.



(Loss over 30k iterations)

I also attempted to create a much deeper RNN with 16 layers instead of four. However, the model could not get below a loss of 3.1 after more than 10k iterations, while other models could get below that after just a couple hundred iterations. These results imply that the model was too complex for the problem. I also attempted a higher learning rate, but this resulted in failed training because some tensors were assigned values that were too big or too small. Finally, I tried doing the four-layer RNN model with 50k iterations instead of just 20k. On top of taking extensively longer, the increase did not create a better model and performed the same as the 20k iteration model.

Conclusion:

The best model tested was 4 RNN and dropout layers, dropping about 20% of the outputs. This model could generate Rock lyrics with good structure and decent vocabulary, even to the point of generating grammatical strings. With this being a good starting point, more

specific hyperparameters could be explored, such as optimizer and loss function. However, since training the models would usually take several hours each, the combinations of hyperparameters were limited. With better computing resources, we could find different hyperparameters more effectively and optimize the model more easily.

Works Cited

DeepAI. "Exploding Gradient Problem Definition." *DeepAI*, https://deepai.org/machine-learning-glossary-and-terms/exploding-gradient-problem. Accessed 14 June 2024.

Marimuthu, Parthiban. "Dropout Regularization in Deep Learning." *Analytics Vidhya*, 7 November 2023, https://www.analyticsvidhya.com/blog/2022/08/dropout-regularization-in-deep-learning/. Accessed 14 June 2024.

PyTorch Contributors. "RNN — PyTorch 2.3 documentation." *PyTorch*, https://pytorch.org/docs/stable/generated/torch.nn.RNN.html. Accessed 13 June 2024.