**Project 7: Recurrent Networks**

Github: https://github.com/squinton-gcu/Data-Science/tree/main/DSC-550/Project%207

Sophia J Quinton

Grand Canyon University

DSC-550: Neural Networks

Dr. Ali Wahid

23 Feburary 2022

**Project 7: Recurrent Networks**

The architecture for the min-char-rnn.py is a simple recurrent network. A recurrent network is a type of neural network that will take into consideration the results from the previous step and the current step to make a decision (Laskowski, 2021). Recurrent neural networks are often used for speech recognition and natural language processing. There are a few different types of RNNs available.

The differences between the three outputs are the loss function and the total number of iterations. For this step I needed to determine what an epoch was in this context. An epoch is considered one whole pass through the entire dataset regardless of iterations and batches (Baeldung, 2021). In this context, 25 steps are the batch size. There are 1115394 total characters in the dataset. I altered the while loop to take into consideration the number of runs through the tiny Shakespeare dataset (Karpathy, 2016). For epoch 5, the data set ran about 1 hour. The final loss was 51.7 and at iteration 2230000. From the gradient check analysis, the relative error was around 1.96e-10. This is below the threshold of 1e-5, which shows that the gradient descent is working for the model. For 50 epochs, the dataset ran for 3.5 hours. The final loss was 47.52 and 2230700 iterations. The relative error for 50 epoch was also around 1.83e-10. For 500 epochs, the runtime was about 16 hours for 282 epochs. I was unable to complete the 500 epochs before the deadline of the assignment, but at this rate it would have been about 24 hours to complete this analysis because the dataset is large. The relative error was at 7.9e-10 and the loss was 38.2 for 12539600 iterations. The general trend that was noticed was that the loss decreased with the number of iterations, but the relative error of the gradient dropped then increased. This is probably due to the issue of overfitting with so many epochs. The loss function does not decrease as rapidly as it does in the beginning of the analysis.

The gradient check allows the user to determine if the analysis is below threshold. Some challenges of the basic RNN are that it is only dependent on the previous result for the result of the current set. This is a limitation to the learning process. I also noticed that the loss function seems to plateau at some point in about 20 epochs. I am wondering if this is because this network does not have any memory to the analysis. The biggest problem I encountered is the runtime of the program.

Overall, I think there would be a benefit to using one of the RNN modification algorithms that allow for more of a memory-based structure. This would help decrease the loss function sooner in the analysis. This vanilla RNN model has several limitations that decrease speed and accuracy.

For the outputs and code, it has been uploaded as separate documents. The code for the notebooks and the raw code has been uploaded to Github for convivence.

**References**

Baeldung. (2021, February 27). *Epoch in neural networks*. Baeldung on Computer Science. Retrieved February 23, 2022, from https://www.baeldung.com/cs/epoch-neural-networks

Karpathy, A. (2016). Char-rnn. [Source code]. https://github.com/karpathy/char-rnn

Laskowski, N. (2021, July 12). *What are recurrent neural networks and how do they work?* SearchEnterpriseAI. Retrieved February 23, 2022, from https://www.techtarget.com/searchenterpriseai/definition/recurrent-neural-networks