# Deep Learning A4, LSTM RNN

Mark Ward
Center for Data Science
New York University
maw627@nyu.edu

## 1. Overview

This assignment studied Long Short Term Memory Recurrent Neural Networks (LSTM RNN) [1]. We learned about the inner workings of the model and its architecture. We also built on top of the model to be able to finish sentences one word at a time. Finally, the LSTM RNN network was altered to produce a language model at the character level. The dataset used was the Penn Treebank dataset.

## 2. Models

The LSTM RNN network that was built was based on the code from Wojciech Zaremba and made use of the nngraph package. The core component of the model is the LSTM cell, which is where the model is able to access information from previous states, its memory. Figure 1 shows the connections within a single LSTM cell, an arbitrary number of these may be stacked one on top of the other. The architecture of the LSTM cell used in this model follows the description in Zaremba 2014 [2] exactly.

I had experimented with multiple parameters to test out different models for language modeling at both the word and character levels. Changes to the model included adding more layers (LSTM cells) and altering the size of the initial embedding for a word or character. I believed that the model for words may benefit from a higher dimensional embedding since each individual word may contain a lot meaning that could be captured more easily in a high dimensional space. On the other hand, individual characters do not contain a lot of information by them self but only in larger sequences of characters. For this reason I believed that a character model may benefit from more layers and a smaller dimension vector embedding.

## 3. Training and Optimization

I had good results using the parameters very similar to what was given as the default baseline but I found it very difficult to train larger models as the number of model parameters grew very large, due to more layers and/or higher dimensional embeddings. Training was done one a training set and there was also a validation set that was reserved to evaluate model performance. Since the dataset was not very large, I used dropout with probability $0.5$. I used batch gradient descent with a batch size of $20$. The learning rate was initially set to $0.9$ and stayed fixed for $5$ epochs and then was halved after each epoch there after. The reason the learning rate was set so large was because the gradients were clipped. The gradient vector had a maximum norm of $5$ and any gradient that had a norm that was greater was scaled down to the appropritate norm but the direction was unchanged.

## 4. Questions

1. See the code in nngraph_handin.lua

2. i $= h_t^{l-1}$, prev_h $= h_{t-1}^l$, and prev_c $= c_{t-1}^l$

3. The function create_network() returns a single network that was created using nngraph and is of type nn.gModule. The network is not unrolled, that occurs when we call the clone many times function.

4. model.s is used to save the state of the RNN between time steps and between layers. model.ds is used during the backprob to store the gradient and model.start_s is used to reinitialize model.s[0]. It gets reset after each pass over all the training data.

5. The gradients are clipped. There is an option to set the max norm of the gradient vector and if the norm is greater than this the gradient is scaled to the max allowed norm.

6. Batch gradient descent is used, without momentum.

7. For the backward pass I pass an additional tensor of all zeros that is the same dimensions as the log probabilty tensor that was outputed during the forward pass, batch_size x vocab_size.

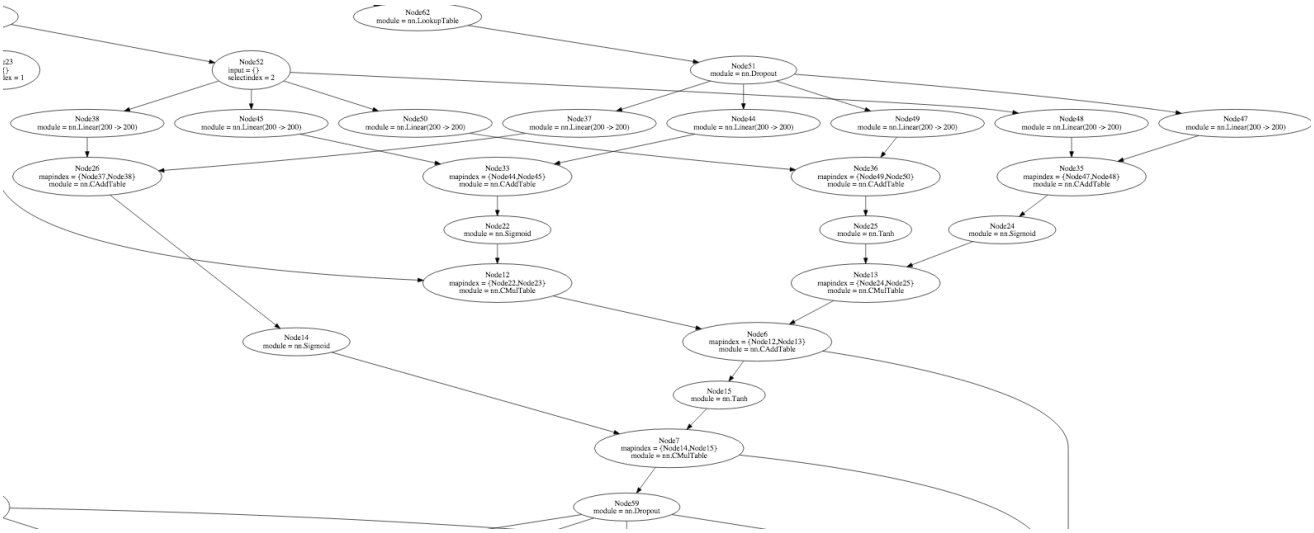


Figure 1. Part of the LSTM RNN network focused on a single layer

## References

[1] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[2] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.