MCSC 6230G Advanced Topics in High-Performance Computing: Assignment 3

Luisa Rojas-Garcia

University of Ontario Institute of Technology

December 10, 2017

1 Introduction

The goal of this work is to model the English language as closely as humanly possible through the use of Long Short-Term Memory (LSTM) Networks. To achieve this, every word is transformed into a feature vector, which is used as input; and the output is the next most probable word to follow. Consequently, the network will produce a sequences of most probable words, that resemble a coherent English sentence. The network will learn these probabilities through the use of a large list of words, or more specifically a literature novel. The architecture, optimizations, and analyses of the proposed LSTM model will be discussed in detail throughout the paper.

2 Background

In this section, we will discuss Recurrent Neural Networks, their applications and some of their drawbacks. Then, we will examine Long-Short Term Memory Networks as a way to tackle some of the problems.

2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are applied to the analysis of sequential information. As opposed to other kinds of neural networks that handle each input and output independently, an RNN allows older information to linger. This concept is displayed in Figure 1. Given a sequence of data, an RNN is able to predict the output(s) to follow by taking it's output back. This way, the network can compute a hypothesis h based on previous collected knowledge. This can be a very powerful characteristic.

The network handles past data fed to it using a hidden state s [4] (see Figure 1), which can be abstracted as the *memory* of the network, and is calculated based on previous inputs. So, at a time t:

$$s_t = f(U_{x_t} + W_{s_{t-1}})$$

where x_t and o_t are the input and output at step t, and s_t is the hidden state at step t. Typically, this state this calculated using a non-linearity such as $tanh$ or $ReLU$, while $softmax$ is more common to be used in the output ($o_t = softmax(V_{s_t})$.

Unfortunately, RNNs do have some pitfalls. One of them is the problem of **Vanishing Gradients**. As the data moves along different neural networks, it also passes through many multiplication stages; this is necessary in order for the model to establish a connection between an output and data seen several steps before [1]. In this process, it's possible that gradients become too small for a machine to work with. An approach to this problem are Long Short-Term Memory Networks, discussed in the next section.

Nowadays, Recurrent Neural Networks are widely used in a variety of areas. Some applications include language modeling, machine translation, speed recognition and image processing [4].

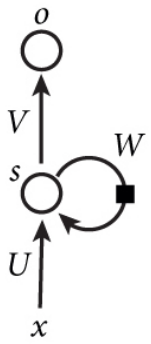

Figure 1: Graphic Representation of a Recurrent Neural Network [4]

2.2 Long Short-Term Memory Networks

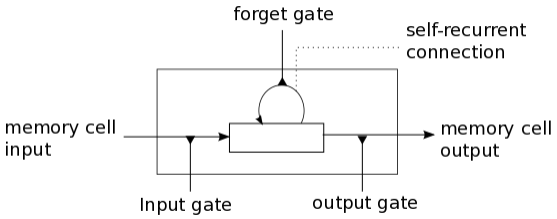

Figure 2: Long Short-Term Memory Cell Structure [3]

Long Short-Term Memory (LSTM) Networks handle the hidden state slightly differently than regular RNNs, but are also capable of learning long-term links [7]. The repeating module s (namely, the *memory cell*) now has a different structure: Through a set of internal neural networks and several gates, this element is able to determine what information to store, and what to forget. This data flow can be visualized in Figure 2.

3 Generating Text with LSTMs

Starting with a base implementation of a Long Short-Term Memory Network, different potential optimizations were applied. The different models were developed using the keras library, and trained on a server with a single GeForce GTX 1080 Ti graphics card.

3.1 Initial Structure

First, a base LSTM network was implemented [1]. As seen in Figure 3, this model consisted of an Input Layer, followed by a single LSTM with 256 memory units. Then, a Drop Out layer as included with a dropping rate of 20% to help avoid overfitting the model. Finally, and Dense Layer with a *sigmoid* activation. Additionally, I used the RMSprop, the recommended optimizer for Recurrent Neural Networks according to the Keras documentation.

The program supports inputs of varying sizes for seeding. In the case that the sentence provided by a user is longer than the current maximum length used, then it is truncated to fit said set length. On the other hand, if it is not long enough, then the input is padded with random words from the dictionary as needed, where an input of size n will correspond to the last n values of the seed pattern.

Before anything, the data to train on was collected and processed. The textual data, which is used throughout the rest of the implementations, is the book titled "Twelve Short Mystery Stories" by Sapper (Ronald Standish), taken from Project Gutenberg[2]. After filtering out all symbols that were not alphabetical or punctuation marks, the total number of words collected were $100,440$, from which a dictionary of $5,686$ was built.

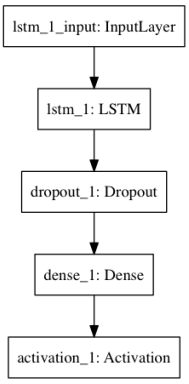

Figure 3: LSTM Implementation, Base Structure

Next, in order to generate the different sentences to train from, I used a maximum length of 15 per sentence and a sliding window of 3 when iterating through the original text. Following this idea, a total of $33,475$ patterns were created; this implies that there were some overlapping words. All sentences, or patterns, are also accompanied by their corresponding target, which is the

[1] This model was created with the help of multiple blog posts and tutorials, including [2].
[2] http://gutenberg.ca

actual word to follow. A few example sentences are shown in Figure 4. These were all, then, converted into one-hot vectors.

```
[pull up just behind her , said ronald . his eyes were ]        [darting]
[, mr . standish not this afternoon , answered ronald . bob ]   [and]
[been in . and the diamonds ain t found them , ]                [sir]
[. yes i knew him , grantham , said ronald . he was ]           [a]
[a pin drop in that crowded hall . for every one of us there ]  [realised]
```

Figure 4: Sample Sentences for Training

For training, I used 100 epochs and only proceeded to save the weights if the loss improved for that specific epoch iteration. In terms of cross entropy loss throughout training, the results are presented in Figure 5. The loss, as expected, decreases over time, but stabilizing after a certain number of iterations (50).

At its lowest, the loss was 1.516754268 at epoch 98. This is a sample output of the model at this point during training, given the seed `"thing would be quite different ; i shouldn t have worried you . but"`:

```
 a a man of the her when out had had getting mrs.  williams, and
was only not as then.  was it s see the i was to my house i went on
went out.  but and a small writing, said ronald.  however, a ve got
a found the very the for as i, certainly said her and is left no.  and
i knew i was better, and when he had us at there in no who happened.
it s which went on with at , at time for time it, an up old with to
there s no very good.
```

The speed at which the loss improves is satisfactory; however, in the next section I will explore alternatives and possible modifications to improve the results.

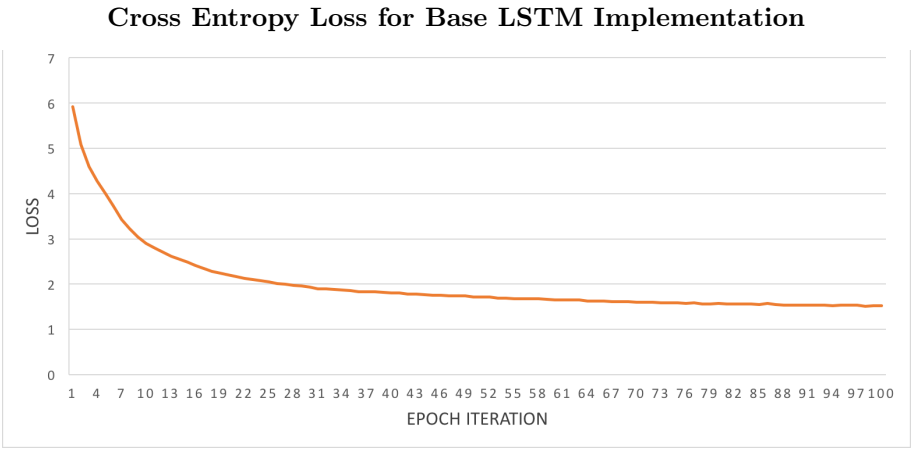

Figure 5: Results for Cross Entropy Loss over Epoch Iterations for the Base LSTM Model Implementation

3.2 Modifications

A number of possible additions to the model were considered in order to find improvements from the LSTM. Each of them will be talked about in detail in the sections below.

3.2.1 Length of Training Patters

The first approach to improve the current model will be by varying the length of each sequence in the training set. I changed the length of each pattern from 15 to 5, and adjusted the sliding window accordingly, from 3 to 2. With this change, the number of training sets increased significantly from $33,475$ to $50,218$. The results for this experiment are shown in Figure 6.

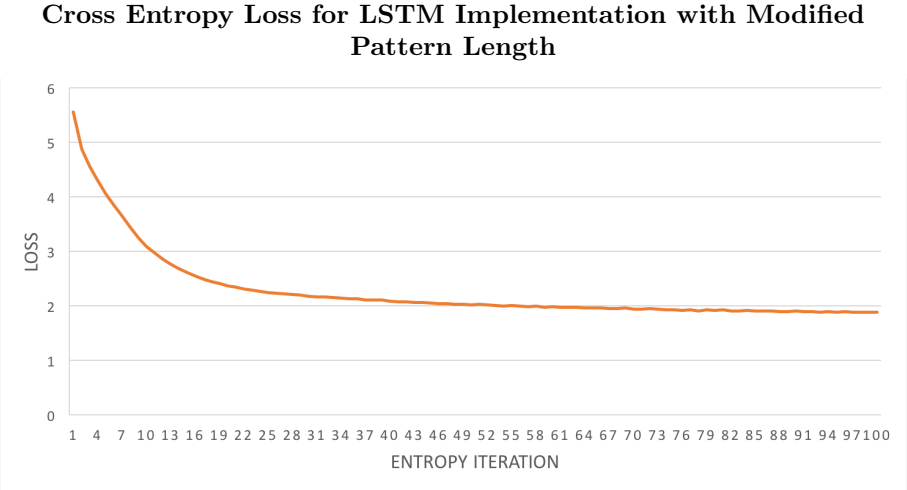

Figure 6: Results for Cross Entropy Loss over Epoch Iterations with Length of 5 for All Training Patterns.

The outcome did not surpass that of the initial model. Though the loss decreases relatively quickly, it's lowest only reaches 1.881628014 at epoch 98, as well. For this reason, the length of 15 will be kept for future modifications.

Moreover, it was the case for both outcomes that the loss did not decrease by much after epoch 50. Due to this, in addition to limitations in terms of hardware, the next experiments will be carried out using 50 epochs.

3.2.2 Second LSTM Layer

The second approach was to add a second LSTM layer with 256 memory units as well. Following this layer, a second Drop Out Layer with a drop out rate of 20% was also added to accompany it. The results, displayed in Figure 7, did not improve.

Even though it did decrease slightly after epoch 8, it wasn't much. The loss was essentially constant throughout the iterations; there wasn't much change from the value it started with to its lowest: 6.354732551. This may be too complex of a model for this kind of problem; more than one layer is likely not needed.

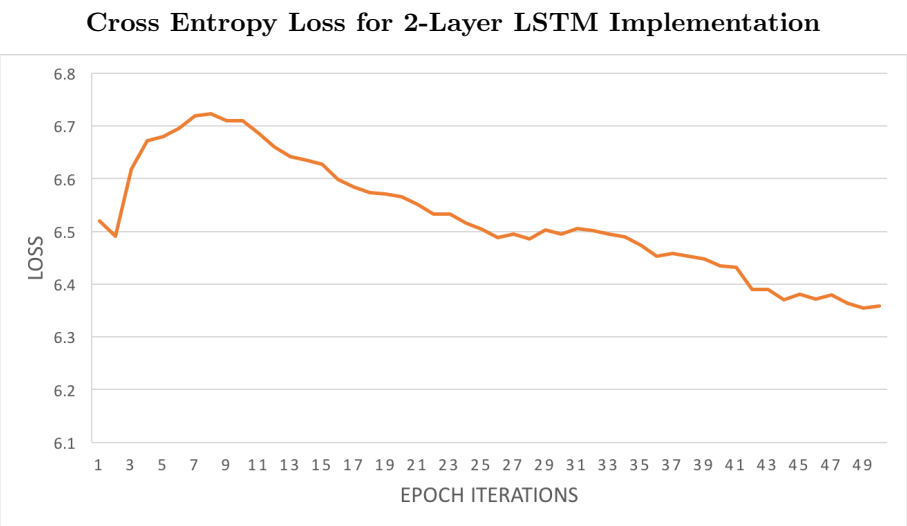

Figure 7: Results for Cross Entropy Loss over Epoch Iterations with a second LSTM Layer and Drop Out Layer

3.2.3 Prediction Diversity

According to Karpathy [6], another possibility is altering the **temperature** of the $softmax$ during sampling, where decreasing its value from 1 increments the confidence and conservatism of the network, and a higher value than 1 will provide more diverse predictions.

More concretely, the temperature is a hyperparameter of Long Short-Term Memory networks used to vary the randomness of it's predicted values. When using this value as a way to alter the predictions, by either constraining them or relaxing them, said predictions are calculated using the following equation [5]:

$$q_i = \frac{exp(\frac{z_i}{T})}{\sum_j exp(\frac{z_j}{T})}$$

For classification purposes, networks output a set of probabilities for each class, which is represented by z, whre $z = (z_1, z_2, ..., z_n)$. q_i corresponds to the new set of probabilities given a value of temperature T.

After implementing this approach, the loss results did not improve. The values for temperature used were 0.5 and 1.5, which returned outputs of minimum loss of 2.0357939865413575 and 2.0455685554237615, respectively. The iteration for these results was at epoch 48 in both cases.

4 Conclusion

The ultimate objective of this project was to generate sequences of words using Long Term-Short Memory networks. To train it, the classic book "Twelve Short Mystery Stories" by Sapper (Ronald Standish) was used, from which $100,440$ words were collected for training. After setting up a model with a single LSTM

layer, a series of approaches were explored in hopes to further improve the model.

Unfortunately, the results for the proposed methods were not satisfactory; the initial model created outperformed all other approaches for potential improvements. It's important to note that a number of these parameters may work differently with other training sets. For future work, I would attempt these methods once again with different data.

Finally, of course, the more data there is available for training, the more variability there will be available in the vocabulary, which is bound to generate better results.

References

[1] D. L. 4J. A beginner's guide to recurrent networks and lstms, 2017.

[2] J. Brownlee. Text generation with lstm recurrent neural networks in python with keras, 08 2016.

[3] DeepLearning.NET. Lstm networks for sentiment analysis.

[4] W. Denny Britz. Recurrent neural networks tutorial, part 1 – introduction to rnns, 09 2015.

[5] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[6] A. Karpathy. The unreasonable efectiveness of recurrent neural networks, 05 2015.

[7] C. Olah. Understanding lstm networks, 08 2015.