Ling 334 Final Project: RNN vs LSTM on a Character Level Task

Gustavo Lucas de Carvalho
Linguistics / Northwestern
Gustavocarvalho2023@u.northwestern.edu

Abstract

LSTM architectures have been presented as superior versions of the normal RNN architecture. This final project explores that by implementing a LSTM and a RNN to the same task, then comparing the performance of both with different sets of hyper parameters.

1 Introduction

Interpreting data in a sequence is a very important task simply due to the fact that a lot of useful information can only be accessed in sequences. A categorical example of this is language. That being said, there are many models that were developed to handle sequence data; namely the Recurrent Neural Network (Rumelhart et al., 1985) and the Long Short-Term Memory architectures (Schmidhuber et al., 1997). These architectures, however, are not created equal; after all, LSTMs were created as an improvement on the RNN architecture. Yet, it is unclear whether the extra machinery in an LSTM would provide a significant improvement from an RNN when it comes to tasks that only include short sequences (such as character level tasks). This project will evaluate exactly that by implementing both architectures at the task of predicting the language of origin of a name.

2 Background

Recurrent Neural Networks, or RNNs, are a kind of neural network architecture that specializes in learning sequence data. By producing a hidden state along with an output, a RNN can keep track of previous inputs and therefore better handle sequence data. To illustrate this, consider the task that is the subject of this paper: predicting the language origin of a name. Given a name, a RNN would receive as input the first letter of said name and output some output vector as well as a hidden state. That hidden state is then fed into the RNN again as input together with the second letter of the name. This process would repeat until all the name's letters have been fed into the RNN. The output vectors produced by the RNN would then be interpreted to determine what language of origin the RNN predicted.

Although RNNs were designed to handle sequence data, the architecture has a couple of shortcomings, the main one being short term memory. To be succinct, RNNs can only retain information from a few previous time steps, meaning that by the end of a sequence the information the RNN stored regarding the first couple of time steps is negligible. Long Short-Term Memory, or LSTM, is an architecture that was designed to solve this problem. They work in a similar fashion to RNNs except they have a series of mechanisms called gates which can learn what information is and is not important to keep. By doing this, LSTMs don't suffer from short-term memory as much as RNNs.

3 Data

The data used for this project will be a directory[1] acquired in a pytorch tutorial of 18 text files, each containing a bunch of names of a given language origin. The names in this directory have the following language origins: Arabic, Chinese, Czech, Dutch, English, French, German, Greek, Irish, Italian, Japanese, Korean,

[1] https://download.pytorch.org/tutorial/data.zip

Polish, Portuguese, Russian, Scottish, Spanish, and Vietnamese.

4 Methods

A character level RNN and LSTM were implemented for the same task of predicting the language origin of a name. The RNN had two linear layers while the LSTM had five. Both architectures were trained by iterating random training examples through the architecture a certain number of times. Additionally, each neural network was trained and then evaluated with a number of different hyper parameters. Namely, both architectures trained with learning rates of 0.005, 0.01, 0.02 and 0.03, with 100000, 200000 and 300000 random training examples. They were then evaluated by forward propagating 10000 random training examples, measuring the accuracy and generating a confusion matrix.

5 Results

Below are two tables of all the accuracies recorded given a combination of hyper parameters for both architectures. If there is an "X" in the cell, it means that such a combination of hyperparameters was not tested.

Table 1: RNN Accuracy

Number of examples	LR= 0.005	LR= 0.01	LR= 0.02	LR= 0.03
100000	58.5%	58.6%	5.6%	5.8%
200000	63.1%	58.0%	6.0%	5.7%
300000	61.9%	56.9%	5.0%	5.6%

Table 2: LSTM Accuracy

Number of examples	LR= 0.005	LR= 0.01	LR= 0.02	LR= 0.03
100000	50.2%	61.0%	76.2%	80.2%
200000	66.2%	83.1%	88.7%	90.5%
300000	79.8%	87.6%	92.3%	92.7%

Figure 1 and 2 are general confusion matrices which were generated by the architectures when they were trained with 200000 examples and a learning rate of 0.01.

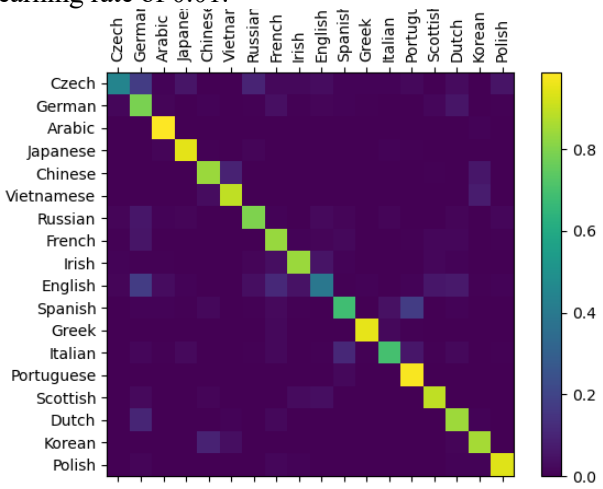

Figure 1: LSTM Confusion Matrix

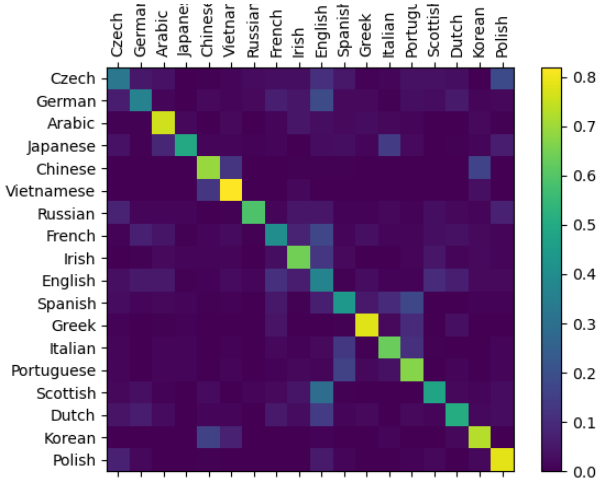

Figure 2: RNN Confusion Matrix

6 Discussion

It is not surprising that the LSTM has outperformed the RNN in almost every combination of hyper parameters. What is interesting is that when the models trained with 100000 examples with a learning rate of 0.005, the RNN outperformed the LSTM by about 8.3%. This is likely due to the fact that, with a learning rate that small, 100000 examples are not enough to properly train the LSTM, which is has more parameters than the RNN.

Another interesting observation from the tables is that the accuracy of the RNN drastically decreased with learning rates of above 0.01. This likely happened due to gradient explosion. It is interesting to note that the same did not happen with the LSTM;

which makes sense since the LSTM gates allows the model to forget information it deemed unnecessary.

While the RNN could not manage to be 70% accurate or more in any combination of hyper parameters, the LSTM was able to be more than 80% accurate in 7 out of 12 hyper parameter combinations (3 of which had an accuracy of more than 90%). Although the LSTM clearly outperformed the RNN, it is unclear whether it would perform this task with this level of accuracy if given a previously unseen data set.

Regarding the confusion matrices of Figure 1 and 2, the diagonal of Figure 1 has a lot more yellow hues than Figure 2, indicating that the LSTM had more true positives than the RNN. Furthermore, there are a lot more non-diagonal squares in Figure 2 that have a green hue compared to the non-diagonal squares of Figure 1, demonstrating that the RNN had more false positives than the LSTM. It is interesting to note that squares representing pairs of languages that have similar roots such as Spanish and Portuguese, English and Irish, as well as Chinese and Korean are some of the most frequent incorrect guesses both models made.

Finally, for future work, the experiment could be repeated but with more data, as to enable the creation of a test set. Although the point of this project was more geared towards comparing the performance of these models to each other instead of the performance on the task itself, it would be interesting to be able to draw conclusions on the proficiency of these models on the task. Yet for that to be possible, a test set with examples not seen in the training data would be needed as to assess the generality of the representations learned by these models.

References

Rumelhart, David E; Hinton, Geoffrey E, and Williams, Ronald J. 1985. *Learning internal representations by error propagation.* Diego, California: Institute for Cognitive Science, University of California.

Schmidhuber, Jurgen; Hochreiter, Sepp. 1997. *Long Short-Term Memory.* Munchen, Germany: Technische Universitat Munchen