Empirical Analysis of Character RNN Implementations in Practice

Justin Nguyen
University of California, San Diego
jhn074@ucsd.edu

Abstract

In the realm of machine learning, recurrent neural networks have emerged to be the definitive option in modeling various applications of natural language processing. As demonstrated in Karpathy's "The Unreasonable Effectiveness of Recurrent Neural Networks", RNNs have shown incredible promise in a variety of applications relating to data structured sequentially. However, Karpathy does not showcase the incredible innovation in recurrent neural network architecture that have gotten them to this point. This paper will seek to take as empirical of an approach as possible in order to demonstrate the differing levels of performance afforded by the most popular approaches of modeling character-level RNNs. Performance will be measured using metrics such as achieved training loss, computational efficiency, and general understanding of generated text.

1. Introduction

Karpathy's blog on "The Unreasonable Effectiveness of Recurrent Neural Networks" demonstrates the versatility of the recurrent neural network architecture, highlighting potential applications in which its effectiveness on sequentially structured data can be leveraged. However, while it synthesizes a strong intuitive understanding of the theory behind the formulation of a basic recurrent neural network, Karpathy's work only provides a rudimentary allusion towards the years of work machine learning engineers have done in order to improve the RNN to level demonstrated in the blog.

The goal of this study is to provide an empirical understanding of the differing performances and advantages provided by well known RNN implementations, ranging from the basic RNN illustrated in Karpathy's blog, the ever popular long-short term memory model (LSTM), and the gated-recurrent unit (GRU). In order to accomplish this, models will be created for each of the implementations and tested with the complete Sherlock Holmes dataset in order to generate text that is characteristic of the novels, and hopefully also of the English language.

Like Karpathy notes in his work, recurrent neural networks still have their shortcomings, especially in the context of character-based text generation. While this study will attempt to use the de facto most performant parameters in order to capture each model at its best, the models will surely exhibit strange behavior that is not characteristic of proper natural language. Nonetheless, these quirks provide another means in which to compare the degree to which each model suffers from RNN shortcomings.

2. Method

This section details the tools used to facilitate this study, including the machine learning framework, RNN implementations, parameters chosen, as well as how performance will be measured.

2.1 Framework, Programming Language, and Hardware

Python: This experiment will be done using Python version 3.7.7 for its high level syntax and numerous machine learning based libraries and frameworks.

Tensorflow-Keras: Tensorflow is a machine learning framework provided by Google and is extremely popular, alongside PyTorch, to be used for applications related to Deep Learning and Neural Networks. I have chosen this framework due to personal preference relating to its ease of use when creating character based datasets, as well as the ability to easily edit layers in order to create the models necessary for this experiment.

Hardware: The experiment will be run locally on a desktop consisting of an AMD Ryzen 7 1700, Nvidia GTX 1060 6GB, and CUDA version 10.1.

2.2 RNN Architectures

Simple RNN: The basic RNN implementation as illustrated in Karpathy's work. This RNN will be comprised of a three layer system composed of an input layer, hidden layer, and output layer that loops in order to generate sequences.

LSTM: The next stage of the evolution of RNN is the Long-Short-Term-Memory model. This architecture features a cell, input gate, output gate, and forget gate that allows the model to learn long term dependencies in longer sequential data. In addition, it also mitigates the problem of vanishing gradients by allowing some information to be passed unchanged. It should be expected for this architecture to outperform the traditional RNN.

GRU: The final architecture to be examined in detail. GRU were created after the LSTM architecture and feature one less gate than the LSTM model. GRUs lack the output gate of the LSTM model and instead use a reset and update gate. In doing this, GRUs pass on the entire cell state to the next iteration. The benefit of such an approach is improved computational efficiency

due to fewer parameters, while maintaining fairly similar approaches. However, this may reduce performance on larger sequences.

2.3 Dataset

Complete Sherlock Holmes: All models will be trained on a large text document comprising the entire Sherlock Holmes collection. This dataset contains 3381928 characters and 97 unique characters decoded in the UTF-8 format. The dataset will be very slightly trimmed by 3376 characters in order to skip the included table of contents and proceed directly to the stories.

2.4 Parameters

In addition to altering the respective RNN architecture of the models, I will also be testing each model with two optimizers. These optimizers are the Adam and RMSprop algorithms as a replacement to traditional stochastic gradient descent. Both of these algorithms use dynamic learning rates and are superior to vanilla SGD. However, both are popular in this application of RNN and merit testing both options for performance. The rest of the parameters, such as those pertaining to the RNNs themselves, will be left constant for the sake of an even comparison among RNN algorithms.

3. Experiment

3.1 Creating Training Samples

The Sherlock Holmes dataset is first converted into a format such that Tensorflow may leverage it as a dataset. Firstly, all characters are transformed into an integer representation corresponding to each unique character. This allows a lookup map to be created in order to retranslate back into character form. Next, this integer dataset is fed into Tensorflow's '.from_tensor_slices()' function in order to separate the dataset into training samples by taking slices as tensors. These slices are then converted into samples by separating each sequence into an input and target sequence that differ by one character, making the task essentially a character prediction task. These sequences are then loaded into batches in order to be used in training the recurrent neural networks.

3.2 Performance by RNN Architecture

Six separate models are created using the Tensorflow-Keras framework. These models are the three RNN architectures: Simple, LSTM, and GRU tested on the two optimizers: Adam and RMSprop. In order to obtain a quantitative value with which to gauge the performance of each model's ability to learn the Sherlock dataset, training loss values will be compared. The training loss in this case will be measured using sparse categorical cross entropy. This accuracy metric is characteristic of categorical tasks consisting of many labels, 97 in this case. Using the sparse variant of categorical cross entropy allows the use of the integer represented dataset, without

having to convert into a one-hot representation which would potentially bolster the spatial complexity of the task. Each model will be trained on 15 epochs.

Furthermore, the average time per training step will be noted for each implementation. This will be the measure with which to compare the computational efficiency of each algorithm. It will often be correlated with the number of parameters to be computed by each approach, but may not always be the case.

Finally, while not exactly empirical, looking at a generated text of size 1000 characters will allow us to observe a visual representation of the learning done by each algorithm. Features to pay attention to may include use of punctuation, sentence structure, and cohesiveness of narrative.

Table 1: Loss and Average time per step

Architecture/Optimizer	Training Loss	Average time per step (ms)
Simple RNN - Adam	1.1164	73.00
LSTM - Adam	0.9294	95.87
GRU - Adam	0.9539	78.47
Simple RNN - RMSprop	1.2876	71.93
LSTM - RMSprop	0.8975	94.73
GRU - RMSprop	0.9238	75.13
GRU + LSTM - Adam	0.8304	301.00

The figure above demonstrates the improvements that the LSTM architecture and the GRU architecture have achieved over the vanilla RNN implementation. When designing the LSTM, improvement to predictive power was the main goal, especially with regard to preventing the vanishing gradient problem. This is clearly illustrated in the table, with LSTM achieving the best training losses of the other two algorithms, no matter the optimizer. The drawback, however, is the heavily impacted computation time as a result of the additional parameters associated with LSTM's input, output, and forget gate. Thus, GRU was developed as a means to improve on computational efficiency, while retaining the longer term memory of its predecessor. This again is illustrated by the figure, showing GRU's much improved computation speed as a result of its one fewer gate, affording less parameters to calculate. While training loss performance is hindered slightly, GRU remains competitive in training loss, while having a computation time that is only slightly worse than a vanilla RNN, which has the fewest parameters.

Table 2: Number of Parameters

Architecture	Number of Parameters
Simple RNN	1,436,001
LSTM	5,371,233
GRU	4,062,561

A model comprising a GRU layer and an LSTM layer was included out of curiosity of the potential results. While it out performed all the previous models, slightly besting the LSTM, it nearly triples the computation time required to train the model.

```
There LNd, I want of his pocket, I pessired bygeal.

     "A very solution."

     "In the colligants, and maid doubt the faces as it was there and something was evidently
     had a most serious rest up, and
     the weight would have had py
     round with a very right had ruch for his mind.
```

Figure 1. Text Generation from Simple RNN

```
There got away into the house," said Holmes. "I wion my friend jealous, as you tell me
     that he and I have seen anything more like a long one to applause. It was
     some story of the middle of it.

     "What is it, then?"

     "Well, you dejected that, there would be all that is better thanks."
```

Figure 2. Text Generation from LSTM

```
There was Mr. Reuland I
     should not have something to decrepatid you will fail to tell you than I.

     "'That with a shrink hat one
     generally is looking at the apologiency quite a bit.
     They appeared.

     Let it threatened by their light her for the tragedy.
```

Figure 3. Text Generation from GRU

Again, while not the most empirical method of observation, there are still some insights gained from looking at the text generated from each of the models. In the case of the simple RNN, the model makes frequent syntactical errors, while also punctuation very simply. We see heavy

improvements in the LSTM model where sentences more resemble English, while also making impressive placement of punctuation - being sure to close dialog with quotation marks, and even mimicking questions. The GRU does worse than the LSTM in this regard, but is still generally better than the simple RNN in syntax. Sentences formed by the GRU still somewhat make grammatical sense, but not to the extent that the LSTM model achieves.

Nonetheless, it is important to note the nonsensical nature of the contents of the generated text, despite initially appearing to be actual novel excerpts. This is still a problem with regards to the char-RNN use case, as previously mentioned in Karpathy's work.

4. Conclusion

The findings in this study highlight the improvement of recurrent neural network architecture via the analysis of the LSTM and GRU. Namely, the improvements in learning and computation speed have important implications in the future of natural language processing. As machine learning continues to innovate, approaches outside of the character-RNN become more effective, such as the encoder-decoder approach to the ever popular transformer model.

I acknowledge some weaknesses of this study, such as using training loss as a quantitative metric for learning performance, as this does not take into account any overfitting that may occur. In creating character-RNNs there are not many methods with which to measure performance, and as such, it was hoped that the size of the dataset would mitigate the effects of overfitting. Furthermore, due to lack of computational resources, it was not feasible to vary every RNN hyperparameter to get the best performance out of every model, thus making more sense to hold parameters constant between every model.

References

The Unreasonable Effectiveness of Recurrent Neural Networks.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

The complete Sherlock Holmes. https://sherlock-holm.es/stories/plain-text/cnus.txt

Sebastian Ruder. (2020, March 20). An overview of gradient descent optimization algorithms.

Sebastian Ruder. Sebastian Ruder.

https://ruder.io/optimizing-gradient-descent/index.html#rmsprop

See Jupyter Notebook for more detailed output