COGS 181 Final Project

Lilyanne Kurth-Yontz

University of California, San Diego

lkurthyo@ucsd.edu

Abstract

This paper serves to highlight the similarities and differences of working with PyTorch and TensorFlow for developing a Char RNN. The shape of the loss curves, the appearance of the final result, and the intuitiveness of the process will determine the strengths and weaknesses of both deep learning frameworks.

I. Introduction

What makes recurrent neural networks (RNNs) so special? According to Andrej Karpathy, author of “The Unreasonable Effectiveness of Recurrent Neural Networks”, RNNs operate over “sequences” of vectors – unlike traditional neural networks and convolutional neural networks (CNNs), which operate over “fixed-size” vectors. As a result, the RNN has more operating power.

Facebook’s PyTorch and Google’s TensorFlow are two of the most popular frameworks used to build and implement neural networks. Both have the potential to generate high-level RNNs. This paper is focused on comparing these frameworks and determining which has the overall best “experience” – what the loss curves look like, what the final result looks like, and what is most intuitive from a programmer’s perspective.

II. Method

The text I chose for the Char RNN was Leo Tolstoy’s “War and Peace”, which contains a little over 3 million characters (almost as long as “Sherlock Holmes”, a recommended text for this project). As for the starting string, I chose “and “, space included. I figured that a phrase this common would generate a wide range of predictions.

For the PyTorch RNN, I derived most of the code from Professor Tu’s extra credit homework assignment, “hw6-rnn”.

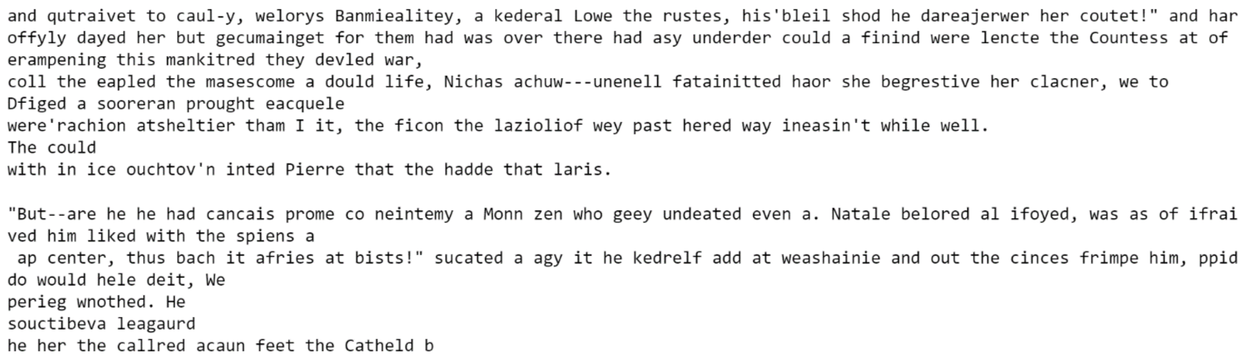
For the TensorFlow (with Keras) RNN, I derived the code from the official TensorFlow tutorial, “Text general with an RNN”, an article by Almis Povilaitis called “Generating Text with TensorFlow 2.0”, and another article by Chuan Li called “Text Generation: Char-RNN Data preparation and TensorFlow implementation”.

I tried to have as many similarities as possible between the two. Both of these models are single-layer RNNs. They take input sequences of 100 characters, and output sequences of 1000 characters.

III. Experiment

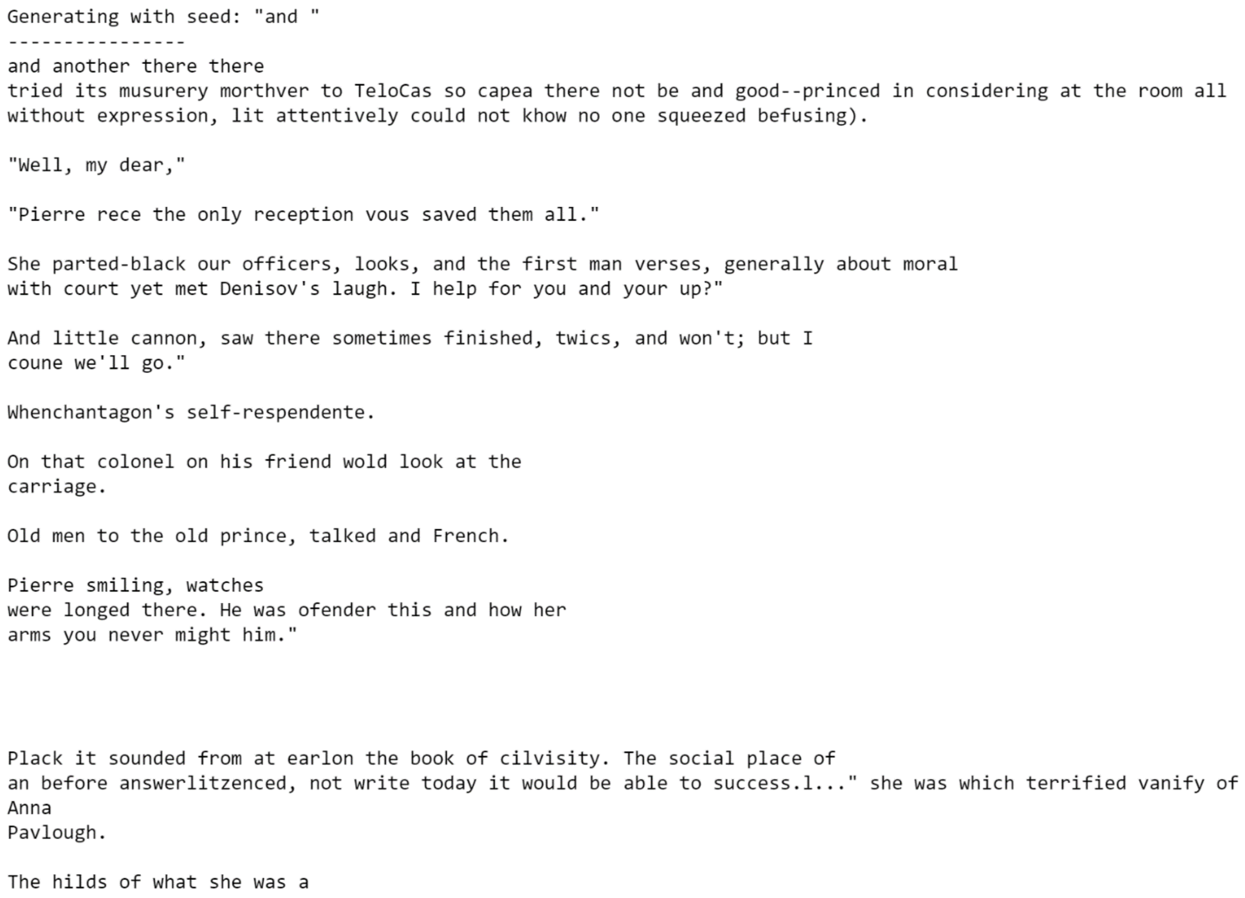
The main question here is – how do PyTorch and TensorFlow compare when it comes to creating a Char RNN? We can look at the final generated text and loss curves, and then we can make a valid assessment.

<u>Text generated by PyTorch</u>



```
and qutraivet to caul-y, welorys Banmiealitey, a kederal Lowe the rustes, his'bleil shod he dareajerwer her coutet!" and har
offyly dayed her but gecumainget for them had was over there had asy underder could a finind were lencte the Countess at of
erampening this mankitred they devled war,
coll the eapled the masescome a dould life, Nichas achuw---unenell fatainitted haor she begrestive her clacner, we to
Dfiged a sooreran prought eacquele
were'rachion atsheltier tham I it, the ficon the lazioliof wey past hered way ineasin't while well.
The could
with in ice ouchtov'n inted Pierre that the hadde that laris.

"But--are he he had cancais prome co neintemy a Monn zen who geey undeated even a. Natale belored al ifoyed, was as of ifrai
ved him liked with the spiens a
 ap center, thus bach it afries at bists!" sucated a agy it he kedrelf add at weashainie and out the cinces frimpe him, ppid
do would hele deit, We
perieg wnothed. He
souctibeva leagaurd
he her the callred acaun feet the Catheld b
```

<u>Text generated by TensorFlow</u>

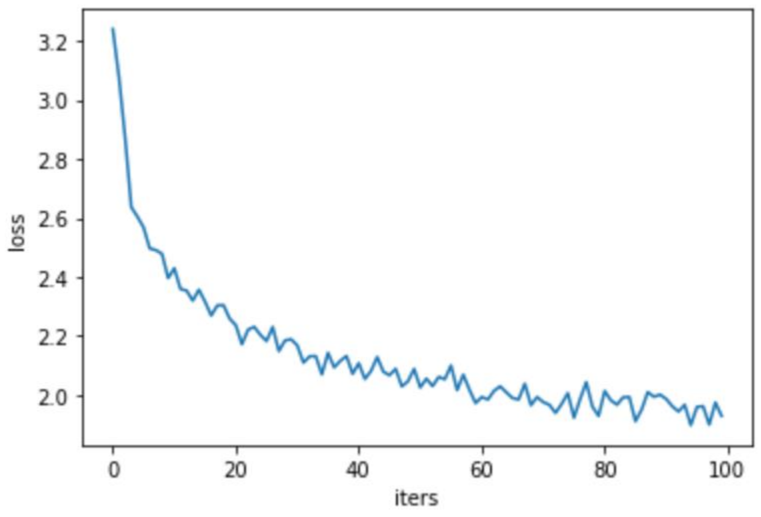


```
Generating with seed: "and "
----------------
and another there there
tried its musurery morthver to TeloCas so capea there not be and good--princed in considering at the room all
without expression, lit attentively could not khow no one squeezed befusing).

"Well, my dear,"

"Pierre rece the only reception vous saved them all."

She parted-black our officers, looks, and the first man verses, generally about moral
with court yet met Denisov's laugh. I help for you and your up?"

And little cannon, saw there sometimes finished, twics, and won't; but I
coune we'll go."

Whenchantagon's self-respendente.

On that colonel on his friend wold look at the
carriage.

Old men to the old prince, talked and French.

Pierre smiling, watches
were longed there. He was ofender this and how her
arms you never might him."



Plack it sounded from at earlon the book of cilvisity. The social place of
an before answerlitzenced, not write today it would be able to success.l..." she was which terrified vanify of
Anna
Pavlough.

The hilds of what she was a
```

Overall, the text generated by TensorFlow looks more polished. The algorithm evidently paid more attention to spacing and how dialogue was structured between characters, and most of the words are actual English. On the other hand, the text generated by PyTorch appears more compressed, and there is considerably more gibberish.

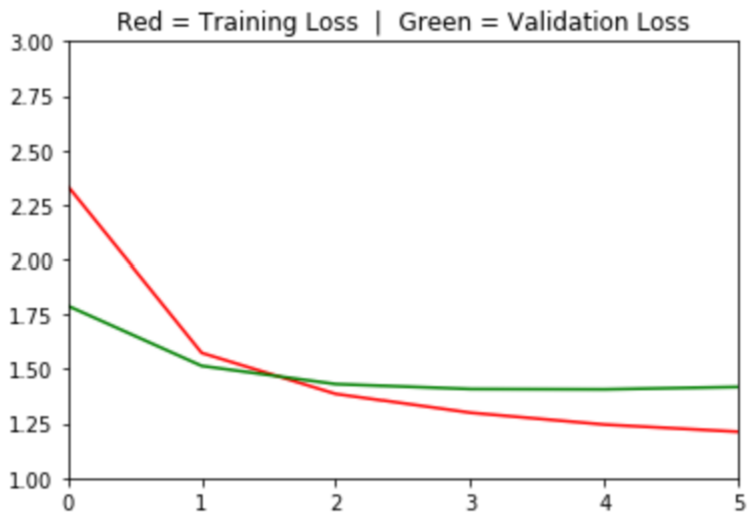
I should mention that PyTorch was unable to iterate all 10,000 times, so I changed the iters parameter to 5,000 instead of 10,000. It would appear that halving the iterations would have a significant effect on the final results, but the loss curve tells a different story.

PyTorch training loss



TensorFlow training and validation loss

(x-axis = epochs, y-axis = loss)



The loss in the PyTorch algorithm, especially the fluctuations toward the end, imply that the algorithm is not really improving. This implies repetitive behavior in the data, as is described in the Google Developers article “Interpreting Loss Curves” (Number 5). The data could have benefitted from some shuffling, like what was done with the TensorFlow data. Because it was a problem with the data itself, adding more iterations would not have solved the problem; we would have kept seeing the fluctuations around the 2.0 mark.

While the TensorFlow algorithm was designed to run for 10 epochs, due to early stopping (in which the training stops when the loss does not improve), it only ran for 6 epochs. As expected by Povilaitis in his article, if training loss decreases for enough time, validation loss begins to increase. This action is pictured in the red training curve and green validation loss curve above.

When it comes to behavior and which framework is more intuitive, the answer is complicated.

I believe PyTorch behaves like a unique implementation, while being easy for beginners to learn. By using a class to build the RNN, PyTorch simplifies the RNN building process; defining a constructor, a forward method, and a hidden state initializer in this way is very familiar for those who come from a traditional computer science background. The iteration process is also simple and straightforward to follow. I would recommend PyTorch for coders from a "vanilla" computer science field who want to build reasonably powerful neural networks.

However, I ran into a very frustrating problem with PyTorch – string.printable was very unforgiving toward texts with special characters. Before using war-and-peace.txt, I used a difference "War and Peace" text file that had letters with accents; the iterations would always fail on the first run. I suspect that the failure of the PyTorch program in iterating 10,000 times was due to finding special characters that weren't entirely eliminated from the text. Regardless, I much preferred TensorFlow's sorted(set(text)), which simply provided a list of unique characters appearing in that text. That way, something as simple as letters with accents wouldn't crash the program.

TensorFlow operates like other, simpler machine learning frameworks. For example, scikit-learn and TensorFlow expect the user to split training and validation sets, reshape and reshuffle them, etc. However, TensorFlow is far from simple – it allows a high degree of control over the neural network process. There were a large number of parameters, but they were defined clearly and could be mapped to specific parts of the code. Plus, it offered nice "graphics", like model.summary() and the real-time progress bar of the training and validation losses. The Keras API also offered many useful utilities, such as extracting text from a URL, that were not innately programmed in PyTorch. I would recommend TensorFlow for coders who are already familiar with machine learning, and who want greater control over the process.

IV. Conclusion

If I could do one thing differently, I would use a LSTM network instead of the "default" RNN. Karpathy describes a LSTM (short for Long Short-Term Memory) network as having a "more powerful upgrade equation and some appealing backpropagation dynamics". An LSTM would likely improve the timing and final text generation of the RNN.

V. References