Erin Kreiling

Group 2

6 December 2018

DATS 6203 – Machine Learning II

Final Project Report

**Introduction**

Frequently, we work on problems in Machine Learning or Data Science where the concern is only about the way our independent variables interact with each other to form an output, or the dependent variable we’re trying to solve for. However, there are also problems in which the biggest predictor is time itself, or at least the time at which actions occur. This project is focused on using long short term memory in a recurrent neural network to answer the following question: do the words of presidents’ State of the Union addresses today rely or even hearken back to the words of their predecessors? Furthermore, do these words evolve over time?

The State of the Union is an annual speech given by the President of the United States, in which they discuss the current state of affairs in the country. This may include the economic outlook for the year, new legislature they would like Congress to be aware of, and an overview of major events of the last year. Since President Woodrow Wilson’s 1913 address, there have been a total of 83 in-person addresses with similar topics but varying tones and sentiments. Research has been done in the past to better understand how the vocabulary and even reading level has shifted over time in these speeches and the results are surprising. The reading level and level of complexity has gradually dropped in the addresses beginning back in the 1700s up through President Trump’s speech this past January, regardless of party affiliation.

The data for this project was pulled from Kaggle and contains the transcripts of each State of the Union address, beginning in 1797 with President John Adams and ending in 2018 with President Donald Trump.

**Analysis**

In order to analyze the text from these transcripts, I selected to use a Long Short Term Memory Unit in a Recurrent Neural Network. These models are typically used when there is some data you want to track long term dependencies in the data, which makes it perfect for this particular problem. However, to use this type of model, there were several steps of pre-processing I needed to complete, which are outlined below.

First, I created a directory with the full list of files to be used for analysis and randomized this list. The purpose of this was to be able to create Train, Test, and Validation sets that each contained data from modern speeches, as well as the older speeches. Due to the nature of the model, it would not be beneficial to create predictions based on only part of the time period. The Training data accounted for 70% of the speeches, the Testing for ~15%, and the Validation for the other ~15%. Each of these datasets was saved in a separate text file.

Next, I began to create the model using the TensorFlow framework. The TensorFlow Mini Project from the Deep Learning Github Repo served as a starting point and was modified to fit the needs of my data. The main changes included selecting a small configuration (with modifications to the individual hyperparameters), changes to the way the methods were called from \_\_init\_\_, and a change to the way in which the loss was calculated. As with most model fitting experiments, these changes were primarily made through trial and error. I experimented a good deal with the number of epochs, batch size, and the hidden size to see which resulted in the most reasonable training time and perplexity. Speaking of perplexity, this is defined as the degree of uncertainty a model has in predicting a piece of text and most of the different iterations produced very low perplexity around 0. This was a particular stumbling point for me in that it took me a few different iterations to identify that the hidden size and vocabulary size needed to be the same, or else the resulting perplexity would be an NA. This caused me to reconsider what I had initialized all of the other parameters to as well and resulted in further tweaking and experimentation. Below are my final selections for the hyperparameters:

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Learning Rate | 0.1 |
| Number of Layers | 1 |
| Number of Steps | 20 |
| Hidden Size | 1000 |
| Number of Epochs | 10 |
| Batch Size | 35 |
| Vocab Size | 1000 |

By using these parameters, the 10th epoch yielded a perplexity of 0.0 and processed 8,419 words per second!

**Conclusion**

In the process of working through this analysis, I had several main takeaways and ideas for future research:

1. Based on this experience, it seems that predicting the next word in a sentence can be very hard, especially if there is no relationship between current words and past words. That being said, a speech given annually by the President of the United States is reviewed by many and future presidents will likely consult those that came before them as they prepare their address. Because of this relationship, this level of comprehension was a bit easier to reach.
2. While phrases or slang change over time, time did not seem to be a huge factor in this analysis. Recent events and trends in politics would be the best indicators for what might be said in the State of the Union each year. For example, in President Trump’s 2018 speech, there is a lot of rhetoric about immigration and this may r may not have been the case in earlier addresses.
3. Effective Natural Language Processing requires huge amounts of cleaning and data munging. While I attempted some, there was far more I could have done that would have yielded a more powerful model. Additionally, the formats and data types used for these sorts of models are absolutely critical. After lots of headache, I realized that the blank lines in my text files were causing issues in my model and proceeded to remove them. This is just an example of how critical it is to ensure that your text data is in the exact format that your model expects and you actually want to analyze.
4. If I were to continue this project, I would test this on the upcoming 2019 State of the Union address (once it is given). I would be curious to know if the model would still have such a low perplexity or if the rhetoric might have changed a bit. Because this is a somewhat turbulent and almost unprecedented political climate, my assumption is that the text could be a bit harder to predict.
5. Given additional time, I would explore more with the impact of changing the number of layers in the model. As with any other hyperparameter, the key is experimenting and weighing the balance between computation time and accuracy.
6. Many of the Github repos and the research I found used Keras as the framework for similar analysis. I would like to try using Keras and see if the performance was drastically different. Given that I know TensorFlow to be a fast framework, I’m not sure that Keras would be better but it would be interesting to try.
7. Similar to point #3, there is a lot more data processing and cleaning that can be done with text data to yield the results you want. For example, I would consider removing stop words and punctuation, as well as performing some checks for misspelling. However, this is exactly what makes text mining so difficult and there is not a set formula to ensuring that you have done the right amount of cleaning and manipulation.
8. Finally, I would prefer to have a deeper understanding of the TensorFlow framework before digging farther into this project. While I learned a lot while digging through the code and trying to make my own adjustments, I feel that I am still lacking some of the main concepts. This also goes for Long Short Term Memory. While I have studied time series analysis in the past, it can be quite different and far more complex when used in machine learning and I would like to research this further to be sure I’m appropriately implementing it.