Presidential RNN

Leon Zhou

Project Design

The aim of this project was to create a character level recurrent neural network to mimic the style and tonality of Donald Trump. This was done by examining a collection of his posted Tweets and official statements as President of the United States. For each corpus, a Markov chain would be generated as a baseline, and multiple RNNs of various depths would be evaluated.

Tools

The primary package used for this project was Keras, using a TensorFlow backend. Training of these models required excessive computation, requiring the use of AWS p2 and p3 instances. Presidential speeches were gathered using BeautifulSoup. Sklearn, Yellowbrick, and Gensim were used to conduct EDA, including topic modeling, clustering, and PCA.

Data

Tweets were obtained from a precompiled database of Donald Trump Tweets. This was a necessity, as the API would not allow tracking of tweets beyond a certain age. Some challenges with this data were that a few rows were not inputted correctly, introducing anomalies in the text. This was handled and removed before training of RNN models, but not before the Markov Chain was already created.

Presidential remarks presented the challenge of separating out Donald Trump's words from that of other world leaders or people present. The notation marking the president's words were not entirely consistent. A complex regular expression was created to extract only the words spoken by Donald Trump, and was tested on a random sample of transcripts. While it succeeded in that context, without going through each transcript individually, it is impossible to say if it was universally correct.

Algorithms

Exploratory analysis initially revolved around topic modeling using LDA. Each corpus was iterated through, using 2 to 20 topics. However, these topics were not clearly differentiable, and contained many ambiguous keywords, such as "thank" and "you." An second approach was to apply PCA to reduce the space to 2 dimensions and apply the DBSCAN clustering algorithm. This produced cleanly defined clusters, and a random sample of the text within each defined cluster did show distinct topics or audiences.

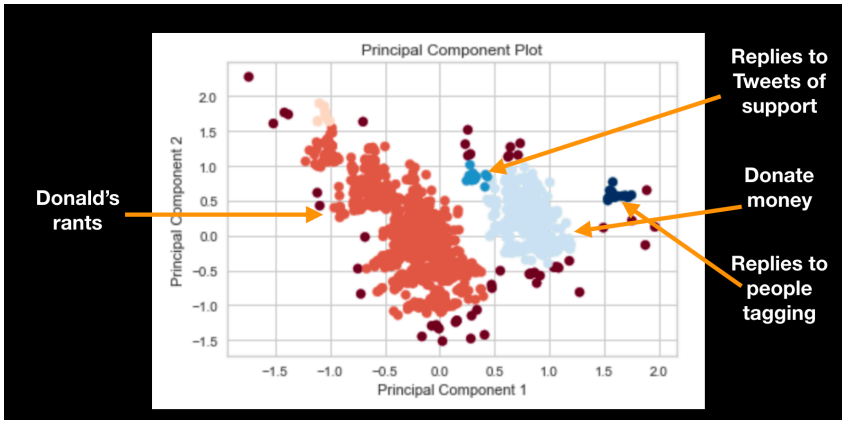

Figure 1. PCA of Tweets, clustered with DBSCAN using eps=0.15, 10 minimum samples.

For the actual text generation, a number of methods were applied. While not really a learning algorithm in the traditional sense, the Markov chain served as an effective baseline of a step-above-random. Beyond that, I utilized a number of RNNs, the most basic of which was a single layer of GRUs, and the most complex surrounding the GRUs with fully connected layers using ELU activation.

```
def rnn_flanked_elu(X, y, bidirectional=False):
    seq_input, x_ohe, output = __init_shared_layers(X, y)

    x = x_ohe(seq_input)
    x = Dense(units=256, activation='elu')(x)
    gru = GRU(units=256, dropout=0.2, recurrent_dropout=0.2)
    if bidirectional:
        x = Bidirectional(gru)(x)
    else:
        x = gru(x)

    x = Dense(units=512, activation='elu')(x)
    x = output(x)
```

Figure 2. Most complex model architecture attempted, with fully-connected layers flanking the recurrent layer.

The GRU was selected as (likely non-comprehensive) literature searches I conducted suggested that performance was typically comparable to that of the more complex LSTM. Given that the GRU had fewer gates, it theoretically would be faster to train than the LSTM, and I believed that any performance hit would be minimal, and worth the time savings.

The ELU (Exponential Linear Unit) activation was chosen after a similar cost-benefit analysis. Most documentation I found suggested that the ELU was slightly more complex, but still comparable to the more traditional ReLU activation function. The ReLU had the potential to experience the vanishing gradient problem, which the ELU claims to solve, speeding up learning and convergence of the model. Any additional calculation of the exponential portion of the activation seemed a worthwhile trade-off.

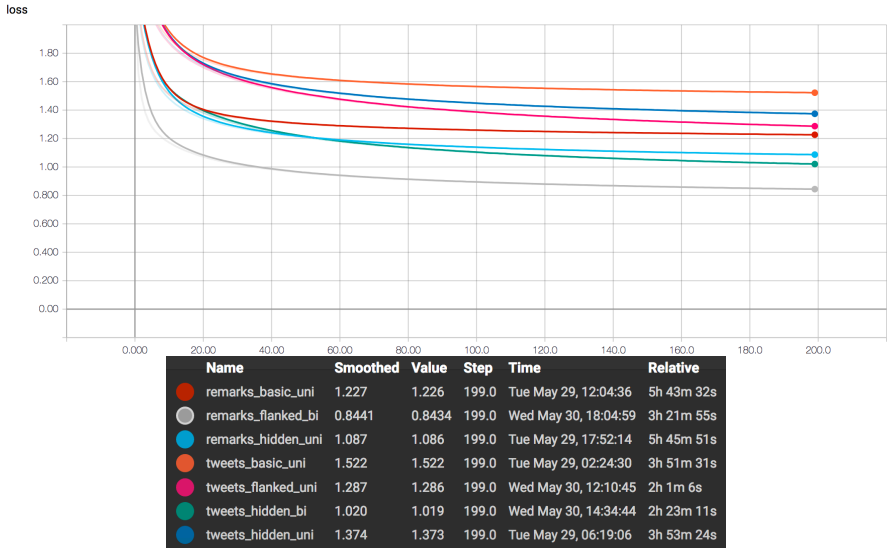

Figure 3. Model loss curves over 200 epochs. Adam optimizer, LR 0.01. Batch size 4096.

Finally, a semi-supervised GAN (Odena 2016)[1] was attempted. Given that most examples of GANs deal with image generation, I was unsure if my implementation would be able to correctly translate the algorithm to a textual domain. In the end, while the code was written, I did not see if this algorithm actually worked, and it unfortunately did not seem an effective use of limited time and computation power.

What I'd Do Differently

Training of my models could have been conducted more efficiently. I started by training each of the unidirectional models for both corpuses before moving onto the bidirectional models. It is likely that the models for one corpus would be equivalent in performance to the models trained on the second, and I should have only trained the unidirectional models on one corpus, before examining the best for bidirectional training on both corpuses.

As mentioned, the Tweet database had some data entry errors that I did not catch until partway through analysis. I had trusted that the pre-constructed database would have data integrity and its columns in order; I should not be so quick to jump to such assumptions in the future. Though, it would likely have proven difficult to catch even if I had spent more time on the preliminary analysis, as the purely textual nature of the data makes it difficult to search for aberrant unique values.

[1] Semi-Supervised Learning with Generative Adversarial Networks
https://arxiv.org/abs/1606.01583