Bryan Richlinski

CS 461: Introduction to AI

Tweet Neural Network Assignment

**Data Preparation:**

I first read the zipped data into a panda file. I then separated each series into its own vector. I then ran to lower on the hashtags and the text data.

I then used the stop word list from the NLTK dataset to remove the stop words from the text body. I then ran scikit-learn’s Snowball Stemmer to lemmatize the text dataset. I then ran sklearn.Tokenizer to extract the top 2000 words, remove punctuation, and turn each word into an integer corresponding to a dictionary. After this I padded all the sequences to match the maximum tweet length.

For the hashtags I used count vectorizer to remove all but the 500 most common hashtags and turn the hashtags into a bag of words with an added NAN category.

I also extracted year, likes, tweet length (by character), and retweets. I used L1 normalization to bring these all between 0 and 1.

As well as this, I also extracted a binary to indicate if there was an @ in the text, this feature didn’t significantly improve performance.

I tried several combinations of features to input to the models to find out what worked and what didn’t.

With a Hashtags, Text Body, Retweets, and Likes the model achieved a test accuracy of 79% but almost immediately overfit. I then tried several combinations of features including:

* Text: 76%
* Hash Only: 72%
* Hash Only but with a size of 900 (Hash 900): 75%
* Hash 900 + Likes: 75%
* Hash 900 + Retweets: 74%
* Hash 900 + Retweets + Likes: 74%
* Hash 900 + Text Length: 75%
* Hash 900 + Text: 80%
* Hash 500 + Text: 79%
* Hash 500 + Text + Likes + Retweets: 79%

I also attempted adding the date feature, this reduced accuracy to nearly random, this may just be because of the normalization step I chose. It may have been better to use a one hot encoding for the year or a numerical range from 0 = min year, max = max year – min year. I did not have time to try this alternative though.

**Network Configuration:**

I settled on using the features Hash 500 and Text. After this I experimented with the number of layers. I simplified the model and started with a model with an embedding layer for the text body data, concatenated with the Hashtag layer, and a single hidden layer of 300 nodes + a dropout layer of .25, and an output dense layer with a sigmoid activation.

Loss: Binary Cross Entropy

Optimizer: Adam

Initial Learning Rate .00001

Diagram

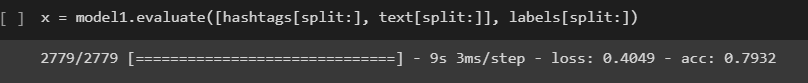
Description automatically generated

In two different runs I attempted to implement a LSTM layer and a 1D CNN layer to the text body data, immediately after the embedding layer. Neither seemed to have a significant impact on the results.

**Validation Strategy:**

I split my data into a train, test, validation split of 70/15/15. Note: I keep the data in one array but use slicing to separate it. I didn’t perform validation on all the models I trained, only the most promising ones. I was performing this all in Colab, which shuts off after a certain amount of inactivity, so training a dozen models for a K-Fold Cross Validation was untenable for me. To lower the risk of overtraining I used callbacks to monitor the test set results every epoch. One that checked the test loss and adjusted the training rate by .1 if the loss values stopped improving for 3+ epochs. I also set up an early stopping callback that stops training after 5 epochs of no improvement to the loss function and restores the best weights. I also included a dropout layer to prevent overtraining as well.

**Results:**



Validation Accuracy: 79%

Validation Loss: 0.4049

I was a little dumb at the beginning and started with a larger network to test the different features I wanted to include. So, I did test this network with 1, 2, and 3 hidden layers with differing numbers of neurons, each with similar points of overfitting. After arriving at the single dense layer of 300 nodes I started manipulating the other hyperparameters.

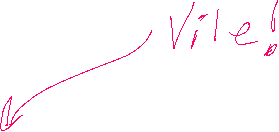
I originally attempted this network with the dropout layer at 25%, a learning rate of .0001, and a batch size of 70. This iteration started high ~78% Test accuracy and almost immediately overfit.

I then adjusted the parameters with a 30% Dropout rate, a .00001 learning rate, and a batch size of 150. This had the effect of starting the model at a lower test accuracy and increasing the accuracy slower, but the result was still 79% accuracy.

**ORIGINAL HYPERPARAMATERS**

Chart, line chart

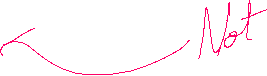
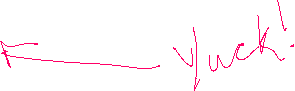
Description automatically generated



Orange = Test, Blue = Train

Chart, line chart

Description automatically generated



**NEW HYPERPARAMATERS**

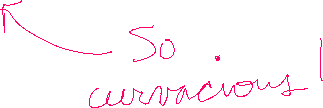
Chart

Description automatically generated



Chart, line chart

Description automatically generated



**Comments**

These results are ok, with approximately 80 accuracy I might trust this algorithm to help recommend twitter accounts but not much more. I did notice that as I increased the number of parameters input into the model from the hashtag set (900 v 500) the accuracy improved marginally. I imagine that if I used a larger vocabulary and hashtag vocabulary my accuracy might improve. But I wonder when the curse of dimensionality would start to take effect. I also suspect that increasing the vector space of the word embeddings may increase the model’s accuracy. I am very surprised though that neither the LSTM or CNN to the model improved accuracy. I suspect that tinkering with these would possibly yield benefit.

As for what went wrong, I think the reason the year ruined my result was because I used L1 normalization, I suspect I could have gotten more information if I had just normalized it to integers in the range of years the tweets occurred, since this was a relatively small range. I also should have started the process with a smaller more basic dataset and built upward instead of the reverse.

**References**

Used the text from my 490 Python Deep Learning class and used assignments from that class as a starting point.