ECE467 Final Project – Sentiment Analysis

Jonathan Lam, Derek Lee, Victor Zhang

May 11, 2021

1 Introduction

We are doing a sentiment analysis task on Twitter tweets. This was inspired by the IEEE 2021 Global Student Challenge[1], in which one challenge is to "analyz[e] sentiments in tweets related to the COVID-19 pandemic."

We chose to use the TensorFlow BERT uncased pre-trained model to perform regression on a numerical sentiment value and classification on classes created by binning those sentiment values. Our model is trained and evaluated on the IEEE "Coronavirus (COVID-19) Geo-Tagged Tweets" dataset[2].

2 Implementation Details

Our code can be found on GitHub at @jlam5555/nlp-sentiment-analysis.

2.1 Dataset

We chose an arbitrary dataset containing tweets related to COVID-19. From the abstract pertaining to this particular dataset:

> This dataset contains IDs and sentiment scores of geo-tagged tweets related to the COVID-19 pandemic. The real-time Twitter feed is monitored for coronavirus-related tweets using 90+ different keywords and hashtags that are commonly used while referencing the pandemic. Complying with Twitter's content redistribution policy, only the tweet IDs are shared.

This dataset contains approximately 360,000 tweets. We did not end up using the geotags, instead focusing only on the tweet contents. (This is actually a subset of a larger dataset[3] containing over 1.2 billion tweets relating to COVID-19, but we only discovered this later, and we figured that the geo-tagged tweets dataset is large enough for our purposes.)

[1] `https://www.computer.org/publications/tech-news/events/global-student-challenge-competition-2021`

[2] `https://ieee-dataport.org/open-access/coronavirus-covid-19-geo-tagged-tweets-dataset`

[3] `https://ieee-dataport.org/open-access/coronavirus-covid-19-tweets-dataset`

As mentioned in the abstract, only the tweet IDs and sentiment labels were provided. The tweets were grouped by day, but we ignored this (only taking into account tweet contents). The tweet contents were fetched using the tweet IDs and the Hydrator tool[4], which uses the Twitter API v2.0 (which requires a Twitter developer account). This returned a JSON file containing all of the approximately 270,000 available (non-hidden and non-deleted) tweets as of April 25th, which forms our main dataset.

The main dataset is this dataset combined with the original labels, with all of the extra metadata trimmed.

2.2 Preprocessing

The ALBERT (A Lite BERT) tokenizer is used for tokenization:

```
input_segments = [
    tf.keras.layers.Input(shape=(), dtype=tf.string, name=ft)
    for ft in sentence_features]

# Tokenize the text to word pieces.
bert_preprocess = hub.load("http://tfhub.dev/tensorflow/albert_en_preprocess/3")
tokenizer = hub.KerasLayer(bert_preprocess.tokenize, name='tokenizer')
segments = [tokenizer(s) for s in input_segments]
```

2.3 Model

The inputs are the hydrated tweet strings (up to 280 characters). We are given sentiment labels, which measure how positive the sentiment of a tweet is. Our main model was composed of an ALBERT tokenizer followed by a BERT encoder.

We performed regression and classification on our dataset. For both of the tasks, we used the main model and appended a few layers on top of it.

For the regression model, we appended a Dropout layer, a BatchNorm layer, and 4 dense layers. For the classification model, we appended a BatchNorm layer, 5 dense layers, and a softmax layer.

We used an 80/20 train/test split. We did not do any hyperparameter tuning, so the test set was also used as the validation set to view the model's performance over the course of training.

2.3.1 Experimentation with network topologies

For regression, our original network comprised 3 dense layers with a single output. Each dense layer used a ReLU activation. The first layer had a width of 128 nodes, the second 64 nodes, and the third 32 nodes (the final layer had 1 node representing the output).

[4] https://github.com/DocNow/hydrator

When training, we realized that the result had almost no variance in the output, so the main goal following this was to try to improve the expressibility of the model's output. Almost all of the values outputted by the model were very close to the mean sentiment value (around 0.11).

We made several attempts to improve the model's expressibility (all to no avail), including:

- Experimenting with the learning rate. One conjecture was that the model wasn't learning fast enough or was converging too quickly, so we tried several builtin optimizers (Adam, SGD), learning rates (the default is 0.001, so we tried 0.01 and 0.1), and learning rate scheduling (e.g., high learning rate at the beginning followed by a lower learning rate, as suggested by this paper).

- The encodings from the BERT encoder may not have enough variance – this may be true if the tweets are considered fairly similar by BERT. We tried using a BatchNorm layer to combat this.

- The model may simply not be large enough. We discovered that the BERT encoding is a 768-length encoding, so even our largest first layer was already constricting the network width. We tried expanding the width and the depth of the layer, trying as wide and deep as a $1024 \times 512 \times 256 \times 128 \times 64 \times 1$ network.

- After examining the distribution of the sentiment labels, we noticed that a very large number of the examples were exactly zero, which likely caused the model to train to roughly a constant value. We tried to combat this by binning the labels into three roughly equal-sized categories and turning the problem into a classification task. However, our results stayed at the baseline levels (i.e., as if we had chosen to most likely category).

- Played around with different activations (e.g., PReLU, ELU), initializers (glorot_uniform, glorot_normal, he_uniform, he_normal) in our desperation.

For classification, our network comprised 4 dense layers with 3 output nodes. Each dense layer used a ReLU activation. The first layer had a width of 512 nodes, the second 256, the third 128, the fourth 64 (the final layer had 3 nodes representing each of the 3 classes).

When training, we encountered similar problems when we were training the regression model. We made similar attempts to improve the model, but the outputs of our model were completely uniform, classifying every input as the most frequent class.

2.3.2 Other challenges

Training time is very slow with the BERT preprocessing and encoder layers. Each training epoch takes roughly 1.5 hours, which means that we were not

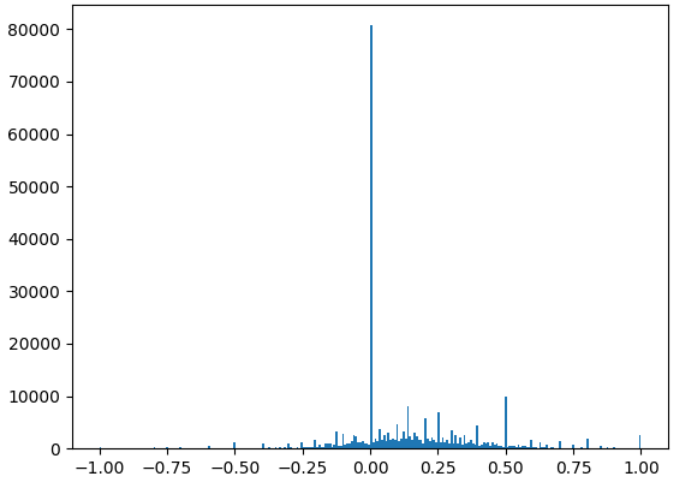

Figure 1: Distribution of the labels. The sentiment labels are on the x-axis. The frequency is on the y-axis. We decided to try binning the labels in order to increase the "variance" of the output.

able to experiment with different models very quickly. Even when all of us were training in parallel, it was hard to make much progress.

Another issue is that, just due to the complexity of TensorFlow, it's difficult to meaningfully "debug" the network. We speculate that our inputs may not be well-suited for the BERT model – even though large neural language models tend to work well on many NLP tasks, our dataset is fairly application-specific. The distribution of the English language in tweets is probably very different than that of the general English language. For example, many of the tweets contained Unicode emojis, which may not have been tokenized properly by the tokenizer.

3 Results

For both the regression and categorization tasks, the best results we got were very close to the baseline. For regression, the best results were similar to the sample mean. For classification, the best results were similar to the accuracy if we tried to classify all inputs as the most frequent class. In other words, our model trained to predict the average label with very little variance, which (probably) indicates that there was a problem with the output of the BERT encoder. (This is the same result that one would get if random inputs were fed into a model.)

For the regression task, the theoretical baseline MSE (calculated over the entire dataset if always predicting the mean value) is 0.0647. With any of our models, we were able to achieve a 0.0644 MSE.

For the classification task, we binned the data into three bins: positive (> 0.05), negative (< -0.05), and neutral. The bins had 33283, 100299, and 135927 examples, respectively; choosing the most probable bin (positive) yields a 50.44% accuracy. We were able to achieve a 50.38% accuracy.

These numbers are close enough to the baseline values that it's fairly obvious that it's predicting the average value (the slight variations are due to train/test splits). When manually observing the predicted values on the test dataset, we see that all of the values are closely clustered around the mean (in the regression case) or are almost all positive (in the classification case).

4 Conclusion

We were not able to achieve state of the art results for the sentiment analysis task. Our model was fairly simple and relied a lot on the BERT encoder and preprocessing TensorFlow pretrained models for the NLP-theoretic part of the project, but we probably implemented the encoder incorrectly. Our experimentation with the network architecture were not able to give us good insight into what our mistake was.

However, since the goal of this project was to learn how to use a deep learning library such as TensorFlow to apply to a NLP task, we were at least able to set up a runnable model and experiment with various model parameters, even if they didn't produce the results we intended to get.