Nicole Joseph
NLP Final Project Report

To run this program: `python .\tweets.py --input1=train.csv`

The Date Set:

The objective of this program is to implement and train a neural net using Python, TensorFlow, and Keras to complete the NLP task of text classification of tweets as disaster (1) or no disaster (0). The dataset used was the NLP Disaster Tweets dataset from Kaggle (https://www.kaggle.com/c/nlp-getting-started). This dataset contains a total of 7,613 tweets, of which 3,271 are marked as disaster and 4,342 are marked as no disaster; this is close to an equal distribution. An example of a tweet marked as 'disaster is as follows: “I know it's a question of interpretation but this is a sign of the apocalypse. I called it https://t.co/my8q1uWIjn.” An example of a tweet marked as no disaster is as follows: “I'm not gonna lie I'm kinda ready to attack my Senior year ??????????” The data, when displayed in excel, had 5 columns marked id, keyword, location, text, and target. The text column related to the tweets and the target column was in regards to their classification as 1 or 0.
80% of the dataset went to training, and 20% went to testing. Of the training data, 10% was allocated to make a validation set.

Preprocessing:

As part of preprocessing the text, URLs were removed using regular expressions. Punctuation was removed: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
Stopwords were also removed using the nltk toolkit. A stop word, by definition, is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

Tokenization:

As part of the tokenization process, the Counter object was used to count unique words and prepare the text for the RNN. The Counter object iterated over each line in the text column and split each line into an array of words. In total, there were 17,971 unique words in this dataset. The keras module was used to tokenize the training data, or vectorize the text corpus by turning each text into a sequence of integers. This led to each word having a unique index after tokenization. Zero padding was utilized to ensure that all sequences have the same length. The maximum number of words in sequence was set to 20 because that length seemed fitting for a tweet. This max length would be set as the input length to the embedding layer of the training model later. Later on, a dictionary was also created, which checked the reversal of indices (key is the index, value is the word).

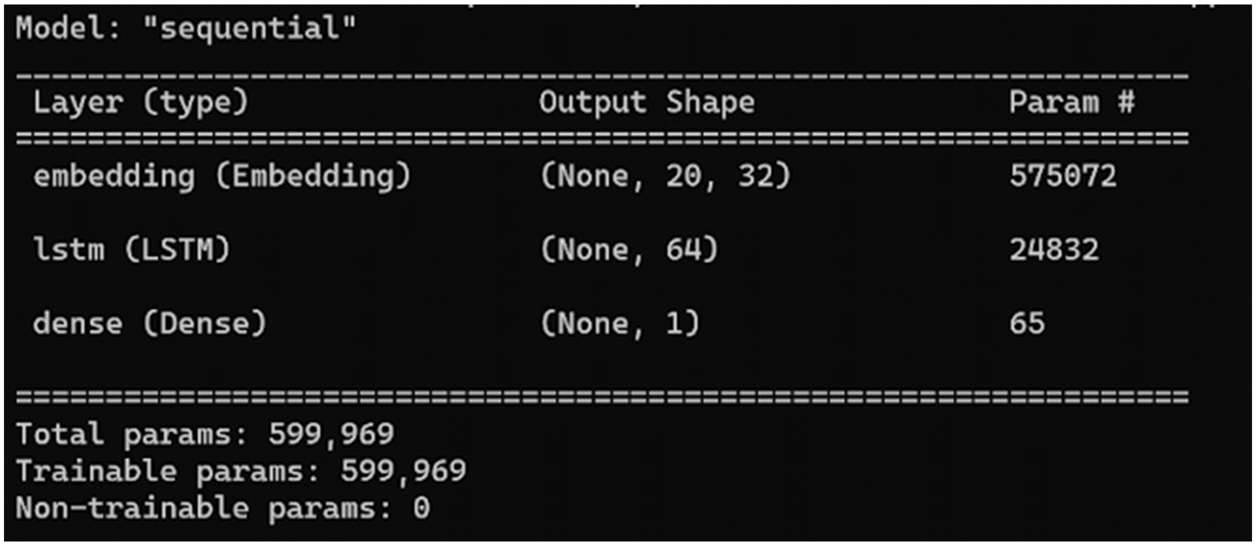
An example of a training sentence to its corresponding sequence and padded sequence:

three people died heat wave far

[520, 8, 395, 156, 297, 411]

[520   8 395 156 297 411   0   0   0   0   0   0   0   0   0   0   0   0   0   0]

Model Architecture:

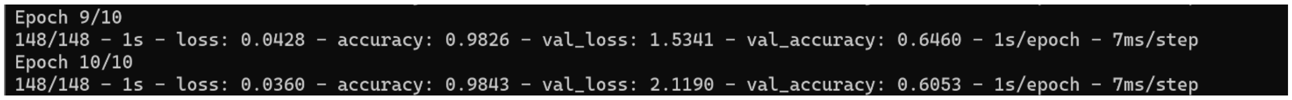


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 20, 32)            575072

 lstm (LSTM)                 (None, 64)                24832

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 599,969
Trainable params: 599,969
Non-trainable params: 0
```

For the machine learning part, a sequential model was set up with imported layers from tensorflow.keras. The model consists of an Embedding layer, LSTM layer, and a Dense layer. In the embedding layer, word embeddings turn positive integers (indexes) into dense vectors of fixed size (other NLP approaches could be one-hot-encoding). Word embeddings provide a way to use an efficient, dense representation in which similar words have a similar encoding. Importantly, this encoding does not have to be specified by hand. The second layer is an LSTM layer. LSTM (Long Short-Term Memory) are a variety of recurrent neural networks (RNNs) capable of learning long-term dependencies, especially in sequence prediction problems. Here, the number of output units and dropout value are specified. The dense layer uses a sigmoid activation function since a 0 or 1 classification is desired. Binary cross entropy function for loss is used since this task is working with binary classification. Also, the adam optimizer is employed.

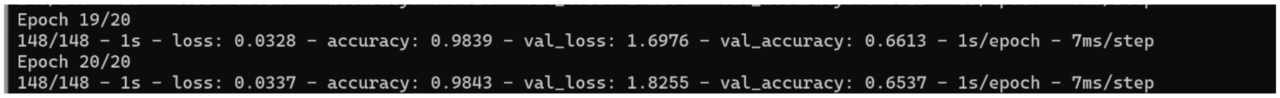
Variation of different parameters:

10 Epochs



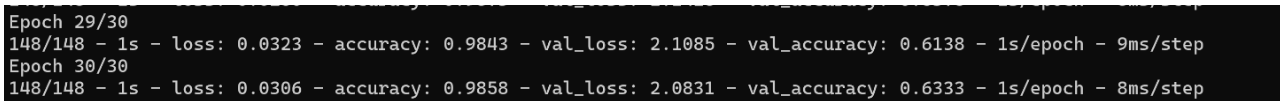
```
Epoch 9/10
148/148 - 1s - loss: 0.0428 - accuracy: 0.9826 - val_loss: 1.5341 - val_accuracy: 0.6460 - 1s/epoch - 7ms/step
Epoch 10/10
148/148 - 1s - loss: 0.0360 - accuracy: 0.9843 - val_loss: 2.1190 - val_accuracy: 0.6053 - 1s/epoch - 7ms/step
```

20 Epochs



```
Epoch 19/20
148/148 - 1s - loss: 0.0328 - accuracy: 0.9839 - val_loss: 1.6976 - val_accuracy: 0.6613 - 1s/epoch - 7ms/step
Epoch 20/20
148/148 - 1s - loss: 0.0337 - accuracy: 0.9843 - val_loss: 1.8255 - val_accuracy: 0.6537 - 1s/epoch - 7ms/step
```

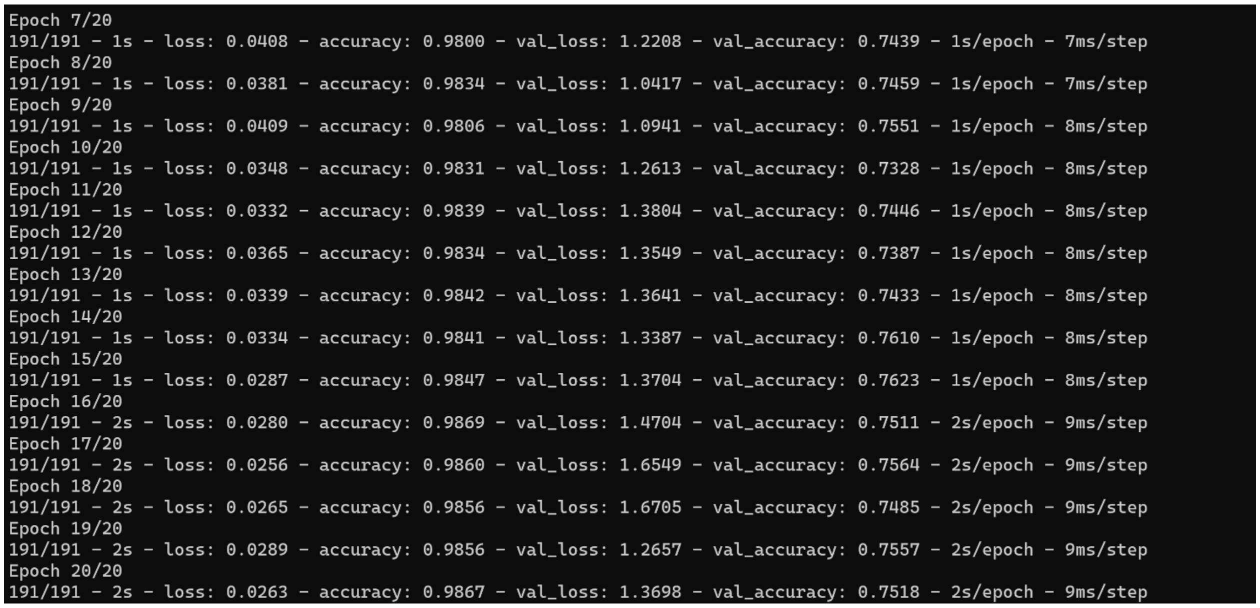
30 Epochs



```
Epoch 29/30
148/148 - 1s - loss: 0.0323 - accuracy: 0.9843 - val_loss: 2.1085 - val_accuracy: 0.6138 - 1s/epoch - 9ms/step
Epoch 30/30
148/148 - 1s - loss: 0.0306 - accuracy: 0.9858 - val_loss: 2.0831 - val_accuracy: 0.6333 - 1s/epoch - 8ms/step
```

When the maximum number of words in a sequence was set to 30, there was a better accuracy value, but 40 worsened the accuracy.

The best combination of parameters yielded an accuracy of 98.67% and a validation accuracy of 75.18%. This is indicative of testing accuracy, which is expected to be lower. The maximum number of words in a sequence is 20, dropout is 0.2, learning rate is 0.1, and the model was trained for 20 epochs to get these results.



```
Epoch 7/20
191/191 - 1s - loss: 0.0408 - accuracy: 0.9800 - val_loss: 1.2208 - val_accuracy: 0.7439 - 1s/epoch - 7ms/step
Epoch 8/20
191/191 - 1s - loss: 0.0381 - accuracy: 0.9834 - val_loss: 1.0417 - val_accuracy: 0.7459 - 1s/epoch - 7ms/step
Epoch 9/20
191/191 - 1s - loss: 0.0409 - accuracy: 0.9806 - val_loss: 1.0941 - val_accuracy: 0.7551 - 1s/epoch - 8ms/step
Epoch 10/20
191/191 - 1s - loss: 0.0348 - accuracy: 0.9831 - val_loss: 1.2613 - val_accuracy: 0.7328 - 1s/epoch - 8ms/step
Epoch 11/20
191/191 - 1s - loss: 0.0332 - accuracy: 0.9839 - val_loss: 1.3804 - val_accuracy: 0.7446 - 1s/epoch - 8ms/step
Epoch 12/20
191/191 - 1s - loss: 0.0365 - accuracy: 0.9834 - val_loss: 1.3549 - val_accuracy: 0.7387 - 1s/epoch - 8ms/step
Epoch 13/20
191/191 - 1s - loss: 0.0339 - accuracy: 0.9842 - val_loss: 1.3641 - val_accuracy: 0.7433 - 1s/epoch - 8ms/step
Epoch 14/20
191/191 - 1s - loss: 0.0334 - accuracy: 0.9841 - val_loss: 1.3387 - val_accuracy: 0.7610 - 1s/epoch - 8ms/step
Epoch 15/20
191/191 - 1s - loss: 0.0287 - accuracy: 0.9847 - val_loss: 1.3704 - val_accuracy: 0.7623 - 1s/epoch - 8ms/step
Epoch 16/20
191/191 - 2s - loss: 0.0280 - accuracy: 0.9869 - val_loss: 1.4704 - val_accuracy: 0.7511 - 2s/epoch - 9ms/step
Epoch 17/20
191/191 - 2s - loss: 0.0256 - accuracy: 0.9860 - val_loss: 1.6549 - val_accuracy: 0.7564 - 2s/epoch - 9ms/step
Epoch 18/20
191/191 - 2s - loss: 0.0265 - accuracy: 0.9856 - val_loss: 1.6705 - val_accuracy: 0.7485 - 2s/epoch - 9ms/step
Epoch 19/20
191/191 - 2s - loss: 0.0289 - accuracy: 0.9856 - val_loss: 1.2657 - val_accuracy: 0.7557 - 2s/epoch - 9ms/step
Epoch 20/20
191/191 - 2s - loss: 0.0263 - accuracy: 0.9867 - val_loss: 1.3698 - val_accuracy: 0.7518 - 2s/epoch - 9ms/step
```

Sources:
https://vgpena.github.io/classifying-tweets-with-keras-and-tensorflow/#so-what-do-neural-nets-do

https://www.kaggle.com/c/nlp-getting-started : NLP Disaster Tweets

https://stackoverflow.com/questions/34293875/how-to-remove-punctuation-marks-from-a-string-in-python-3-x-using-translate/34294022

https://stackoverflow.com/questions/5486337/how-to-remove-stop-words-using-nltk-or-python

https://www.tensorflow.org/text/guide/word_embeddings

https://colah.github.io/posts/2015-08-Understanding-LSTMs/