Problem Formulation

The focus of this project is to utilize natural language processing (NLP) techniques to detect whether the news (on reddit) is fake or real solely based on its title. Nowadays, fake news has be widespread with many trusted media outlets losing their credibility. This is a problem we face in the new age of technology and its separation from reliable sources is essential. The input of this project is a file containing news headers which are labelled referring to whether they are real (0) or fake (1) (I think? I did not see a meta data file). The output will be a system that can predict the probability whether a new article is real or not based on the title, though the initial data is labelled as 1 and 0, the output is the probability of that. Many data mining functions are required for such analysis such as NLP processing steps and various neural nets.

Fully connected neural network

- EPOCHS: The template came with a set value of 20 epochs, however I noticed that the model still had seem to plateau, therefore I decided to first increase the number of epochs to observe a change. I noticed that the fully connected layer performs better with higher number of epochs. There seemed to be a plateau of loss rate around 37 epochs so to avoid the risk of overfitting the data I will not increase this parameter more.
- BATCH SIZE: Batch size defines the number of samples that will be propagated through the network so for example if we have 1000 samples, it go through them in in batches of 100 (an example). The problem is if the batch size is small (even though it will be faster) it might be less accurate the estimate of the gradient. I also decided to increase the number of batch size from 64 to 70 so observe the change in performance.
- DROP OUT: The number that I set for my epochs seemed a bit high so one of the first changes I made to the model was adding a drop out layer after the embedding layer to avoid overfitting (since the output is randomly subsampled), however did not significantly improve model performance. Adding another dropout layer after a dense layer however did slightly improve.
- LAYERS: I began by adding one dense layer and continued to add them to observe any change in performance. It was recommended by literature to increase the number of neurons in each dense layer as you go down so first I started with a dense (200) and added a second dense layer dense (300).
- OPTIMIZERS: I then decided to work on optimizing the model. Leaning rate is a hyperparameter that controls how much change is needed for the model in response to the estimated error each time the model weights are updated. A small learning rate might result in a long training process and a large one may result in learning a not so great set of weights too fast. Since the template came with the Adam optimizer which has a default learning rate of 0.001, I decided to play around with that value setting to smaller and larger values to compare. A smaller learning rate seemed to have performed better but computation took a bit longer. I also implemented different optimization algorthims such as Adagrad and RMSProp but Adam outperformed them.

Recurrent / multi-layer / Bidirectional GRU

- For this model I only have one GRU layer. The recurrent GRU model did not perform well with too many dense layers. I achieved the best performance for this model with one gru layer, and then a dropout layer and then one dense layer, and the final dense layer works bets with the sigmoid activation function. In the gru layer I also set values for the parameter dropout and recurrent_dropout. Dropout is applied to the first operation on the inputs and re-current dropout is applied to the other operation on the recurrent inputs. Out of the three models, single layered GRU performed best with the same parameters set for the model above with the fully connected layer. I did not observe a particular improvement after applying the stop word removal step to my preprocessing. I also played around with max_len and vocab_size and noticed a slight improvement using 6000 for vocab_size and 60 for max_len.

Recurrent / multi-layer / Bidirectional LSTM

- First, I tried an LSTM model with only one LSTM layer and one dense and dropout layer; it outperformed the GRU model. Then I decided to add multiple LSTM layers to make the model deeper, since we are using sequential data, It means that the addition of layers adds levels of abstraction of input observations over time and representing the problem at different time scales. I found that having 2 LSTM layers as well as one dense layer. Anything more than 2 LSTM layer significantly reduced model performance. Then I tried a bi-directional LSTM (obtains representation of a sequence in the forwards and backward direction) model. My bi-directional model performed better than the LSTM with 2 layers, however optimal performance was achieved by only having one bi-directional LSTM layer, when I added a second one, performance significantly dropped.