**Natural Language Processing**Ravindra Bisram and Danny Hong*Spring 2021*Programming Project #3

For this project, Multi Class Text Classification was done with LSTM using TensorFlow 2.0 and Keras on the default datasets (datasets one and two from project 1). For both datasets, the input dataset is shuffled and then 80% is taken as the training data. In the case of dataset 1, the remaining 20% is used as the validation data since test data is readily available to us to experiment with, but for dataset 2, 10% is allocated to validation and 10% to testing, and the split is done randomly with an external Perl script that we had left over from project 1. When invoked, the program will first ask for the path to the training labels file, and then use this path to load in all of the articles (the same way the folders were set up in Project 1). After it completes training, it will request the path to the test labels file (this file should still include labels so that accuracy can be computed). Please note that the program assumes that the TC\_provided folder is in the same directory as itself.

A TensorFlow Keras Sequential model is used to implement LSTM, and starts with an embedding layer, storing one vector per word and converting the sequences of word indices into sequences of vectors (similar words will have similar vectors after training). Next, a bidirectional mapper is used with a LSTM layer, propagating the input forwards and backwards through the layer and concatenating the output to learn long-term dependencies. We also tried stacking multiple hidden LSTM layers but observed the results to be worse than when it was only one layer. At this stage we experimented with different embedding dimensions, trying 32, 64, 128, etc., but saw no change in performance. We then fit it to a dense neural network to do classification and tried the various popular activation functions learned from previous machine learning classes, including sigmoid, ReLU, Leaky ReLU, and Tanh, and after comparing them all for multiple iterations we observed the best results on average for the datasets with regular ReLU, so kept that for our final model. The final layer is another dense layer with a SoftMax activation function which converts the output layers to a probability distribution.

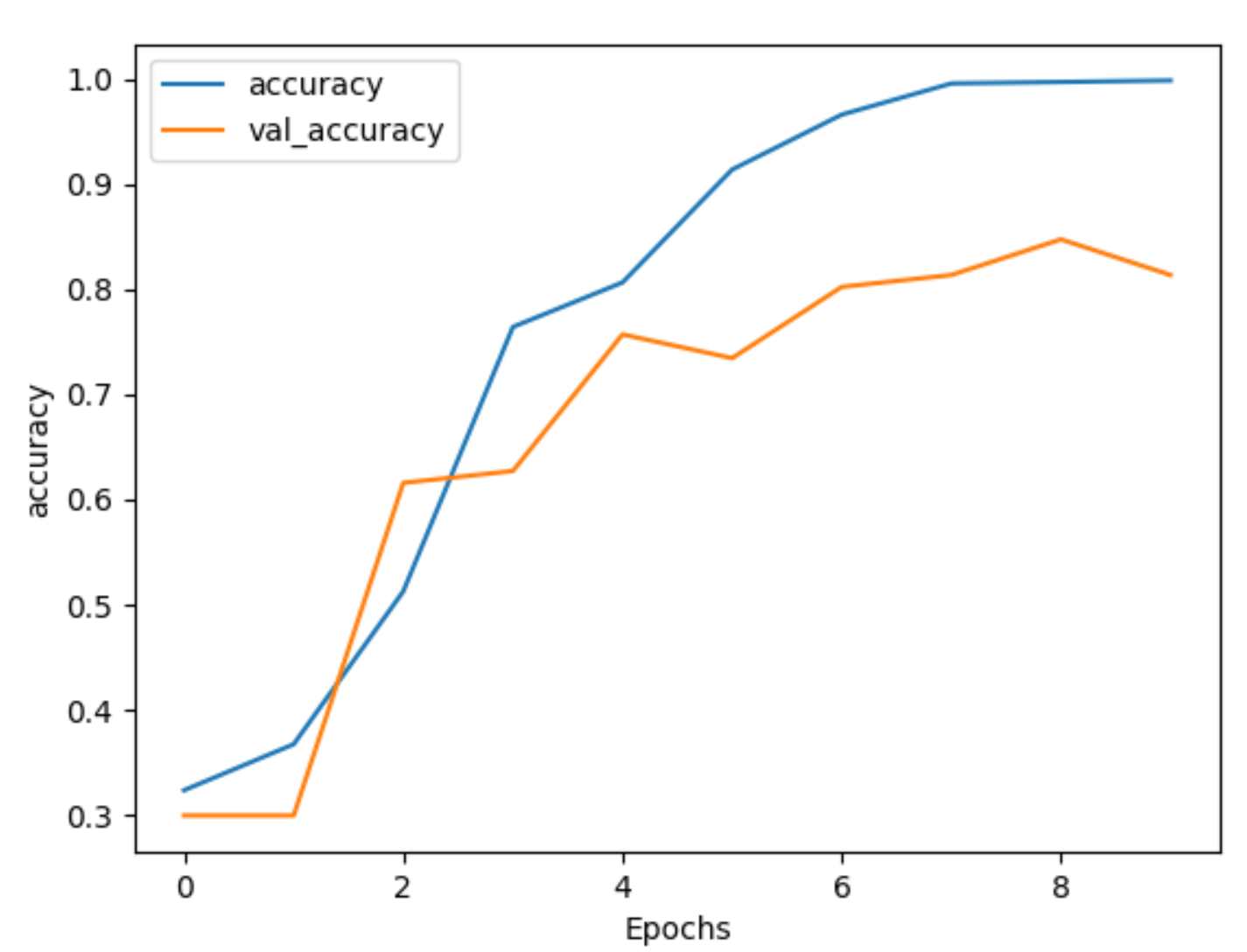
When compiling the model, we had the opportunity to experiment with loss functions and optimization algorithms. For the loss function, sparse\_categorical\_crossentropy was used as it allowed for more than two labels (as is the case in dataset 1) and did not require the labels to be one-hot encoded as would be if we simply used categorical\_crossentropy alone. Since we were playing around with dataset 2 as well, which only had 2 classes, we were also able to experiment with the binary\_crossentropy1 function, but this did not have a significant enough effect to bother implementing different functions and change the number of SoftMax dense layers just for this case. In terms of optimization algorithms, we implemented SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Nadam, and Ftrl. Out of these, SGD, Adadelta, Adagrad, and Ftrl performed terribly for both datasets, consistently resulting in an accuracy at or below 30% for the test sets. On the other hand, RMSProp, Adam, and Nadam all scored relatively well in the mid-90s on accuracy, and we ended up going with Adam out of force of habit more than anything else.

We also played around with parameters that were involved in preparing the data for use in the model, including the vocab size and maximum article length, which actually had little effect on the already well performing model aside from increasing the time it took to run. The vocab size we chose was 5000, meaning that only the 5000 most common words in the training set were added to the vocabulary, and the rest would be assigned ‘OOV’. Of course, the tokenization phase was integral to good performance. We also made use of the NLTK use of ‘stop words’ when processing the data, as these words are unimportant when determining the classification of a text.

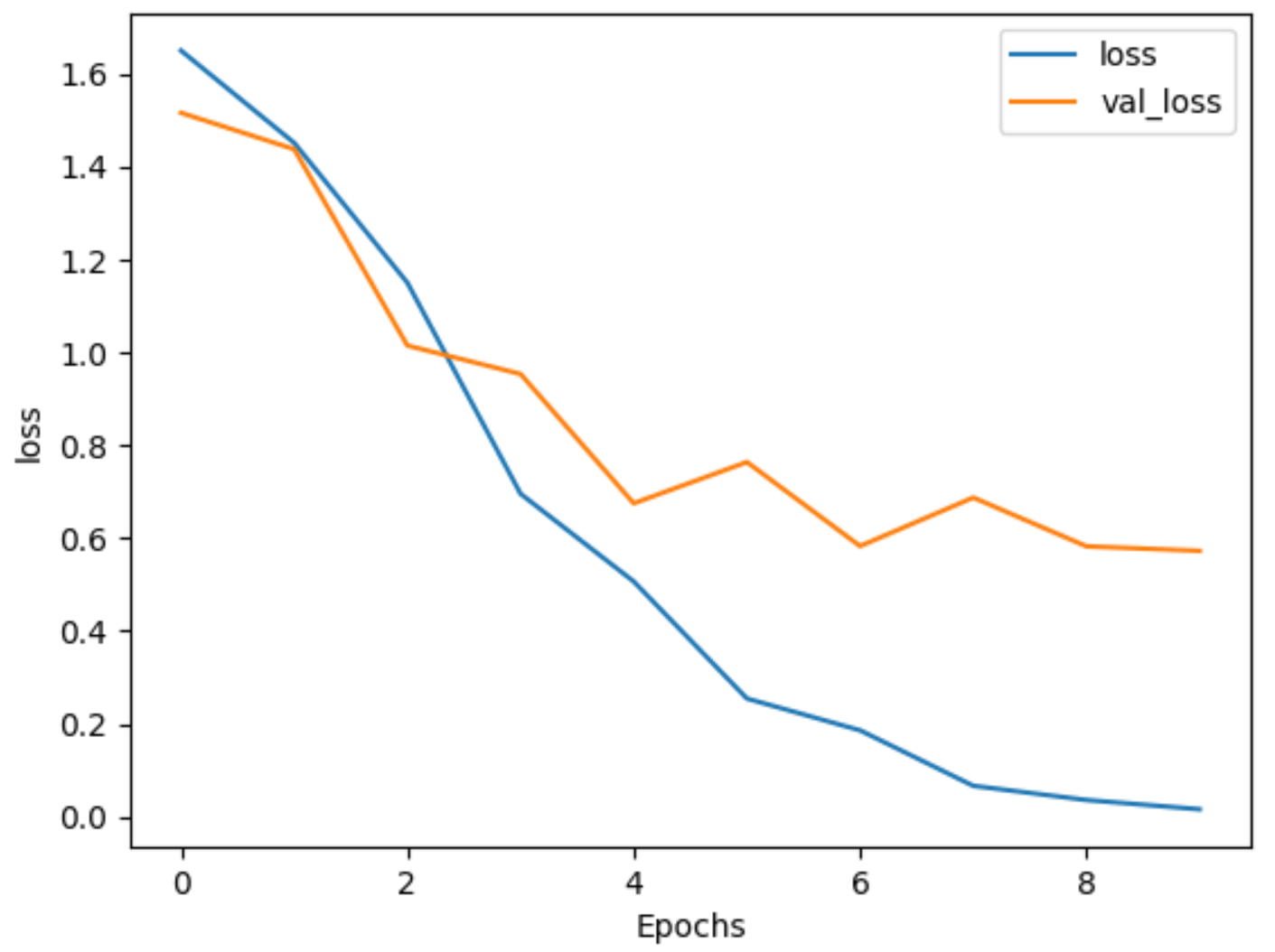
Worth noting is that we are consistently getting a relatively high validation loss with respect to our regular loss, which is an indication of overfitting to the training set. To try and prevent this, we played around with the number of epochs and examined the loss and accuracy over multiple runs to try and find the sweet spot, where accuracy was maximized and validation loss minimized, but this seemed to have no real effect. Too many epochs would also obviously lead to more obvious over fitting, so we chose to go with 10. It was decided that although some overfitting was noticed, it was not resulting in poor test set accuracy results, so we let it slide instead of trying to mess around with the number of layers we had established (this was a possible solution we found in research).

In addition to experimenting with all these factors in this architecture, we had also attempted another similar architecture using TensorFlow that did not pan out nearly as well. Instead of NumPy arrays, this system tried to make use of TensorFlow Datasets which really wasted a lot of time shuffling and selecting and yielded much worse accuracy. This architecture attempted to make use of the BERT natural language model and the HuggingFace pipeline but proved to be unreliable and complicated compared to what we ended up using.

With our final model and parameters, we were able to score on average 95% accuracy for both datasets 1 and 2, which was a noticeable improvement over what we were able to achieve with Naïve Bayes from project 1 (by a couple of points).



**Figure 1:** Accuracy vs Num. Epochs for Dataset 1



**Figure 2:** Loss vs Num. Epochs for Dataset 1

**References:**

<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

<https://keras.io/api/optimizers/>

<https://stackoverflow.com/questions/48479915/what-is-the-preferred-ratio-between-the-vocabulary-size-and-embedding-dimension>

<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

<https://keras.io/api/losses/>

<https://github.com/ralphbrooks/tensorflow-tutorials/blob/master/2-Sentiment-Classification-with-BERT.ipynb>

<https://www.tensorflow.org/tutorials/text/text_classification_rnn>