**D213 Performance Assessment Task 2**

Renzo Espinoza

College of Information Technology, Western Governors University

D213: Advanced Data Analytics

Dr. Festus Elleh

March 15, 2023

**A1:Research Question**

A research question that I will answer using neural network models and natural language processing techniques is: can we accurately predict the sentiment of a user’s review as either positive or negative based on past reviews from other users? The data I will use to perform my analysis is provided by Kotzias et al. (2015), and it is comprised of reviews from the users of three different websites, Amazon, Yelp, and IMDb. The reviews where manually labeled as either positive or negative, making it suitable for performing sentiment analysis.

**A2:Objectives and Goals**

The primary objective of this analysis is to accurately classify a user’s review of a product or service as positive or negative based on their choice of words. With this, we can better understand customer sentiment.

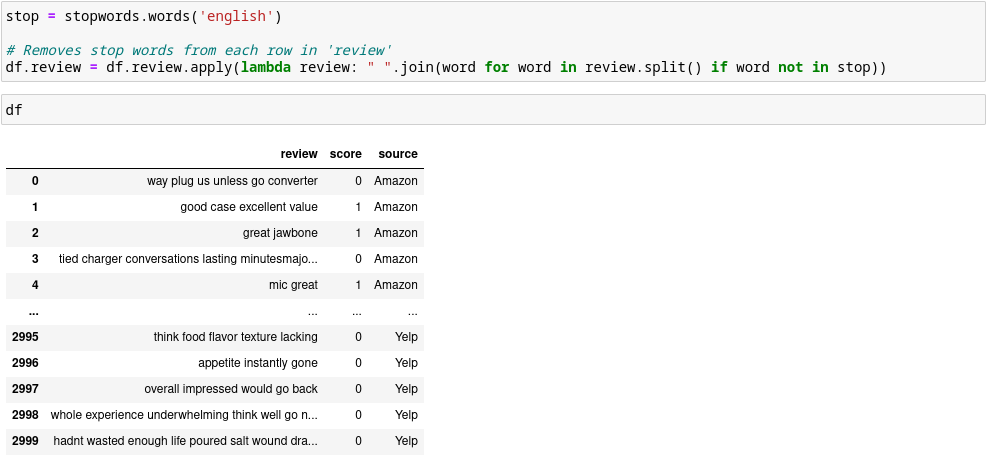
**A3:Prescribed Network**

A type of neural network that is ideal for this scenario are Recurrent Neural Networks. According to Kapoor et al. (2022), RNN’s are frequently used for tasks using text input, such as sentiment analysis, because it works well with any kind of sequential data where the occurrence of an element in a sequence depends on the elements that came before it.

**B1:Data Exploration**

To detect the presence of unusual characters, I first converted all characters to their lower case form to reduce the amount of unique characters without sacrificing much information. Then I iterated through each row, appending each unique character to an array and finally printed the resulting array for review. In the array, there were several non-alphabetical characters that needed cleaning. Now with confirmation of unusual characters, I used a regular expression function to remove all characters that are not letters.

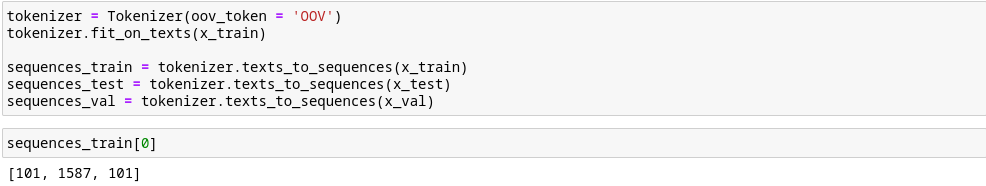
 To remove stop words from each review, I downloaded a package of stop words from the Natural Language Toolkit library and applied a lambda function to each review to remove every word that appears in the list of stop words.

****

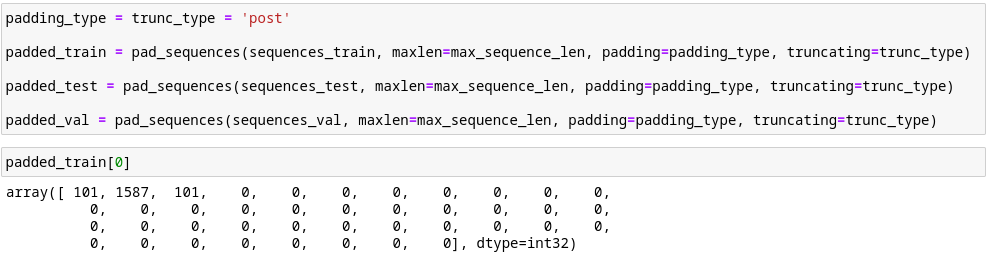
I retrieved the vocabulary length of the dataset by getting the length of the word index list returned from the Tokenizer class in the Keras library. The amount of unique words in my training set is 4181. According to the TensorFlow Team (2017), the word embedding length should be set to the fourth root of the vocabulary size, so I set the length to 8. For the maximum sequence length, Kilinc (2020) recommends setting it to the max amount of words in a sentence, so I set mine to 41. The reason for this is to avoid truncating any sentences.

**B2:Tokenization**

The goal of the tokenization process is to break down each word in a review into a numerical data structure suitable for being processed by the neural network. There are several available tokenizers, the one I used is from the Keras library. The tokenizer succesfully broke down each review into a list of words, then converted them to a list of numerical representations for each word.

**B3:Padding Process**

For the machine to process each of these sequences efficiently, the all need to have a uniform sequence length. I set the maximum sequence length to the max amount of words in any review, so padding the sequences that are shorter than that length is required. I chose post-padding as my type of padding, what this means is that every sequence that is shorter than the max sequence length will be filled with padding tokens until they meet the required length. I used the ‘pad\_sequences’ function from the preprocessing module in the Keras library to perform the sequence padding.

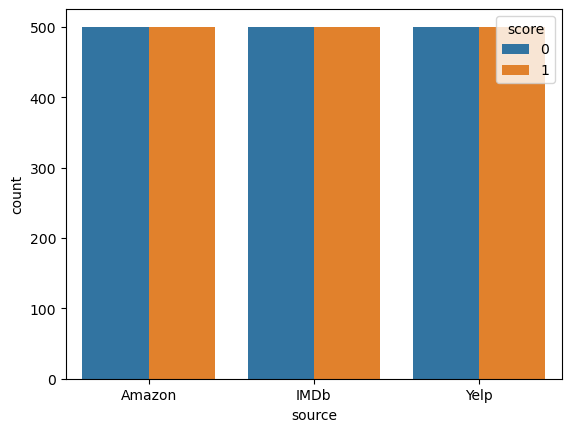
 As mentioned before in section B1, I set the maximum sequence length to the max amount of words in a review to avoid truncating any sequences. The downside to this is you end up with many sequences like the one shown above, which contain a large amount of padding tokens. This can negatively affect the RNN's performance because these zeros can overload the RNN cell's short-term memory and cause it to forget the tokens at the start of the sequence. To avoid this issue I will add masking to the embedding layer.

**B4:Categories of Sentiment**

I will use two categories of sentiment, 0 for negative and 1 for positive. Since there are only two possible outcomes, the activation function for the final dense layer of the neural network will be a sigmoid function as it is commonly used for binary logistic regression.

**B5:Steps to Prepare the Data**

The first step to preparing the data for analysis was to import the three sources of data into my programming environment. I used the ‘read\_csv’ function from the Pandas library, and set tab character ‘\t’ as the separator. This correctly formatted the data into two columns, one for the review itself, and the other for the sentiment of the review. I looked through the data and noticed that reviews starting with quotation marks would cause many consecutive reviews to be treated as one for some odd reason. I manually cleaned several reviews in the three data sources starting with quotation marks and this resulted in all 3,000 reviews being imported correctly.

I then visualized the data to better understand the distribution of positive and negative reviews for Amazon, Yelp, and IMDb. Turned out that there was an even split of positive and negative reviews from each of the sources.

The next step was to look for unusual characters in the dataset. As mentioned before in section B1, I used a regular expression to find and remove all non-alphabetical characters except for empty space. After that I removed every word that appears in the list of stop words provided by the NTLK library from the reviews.

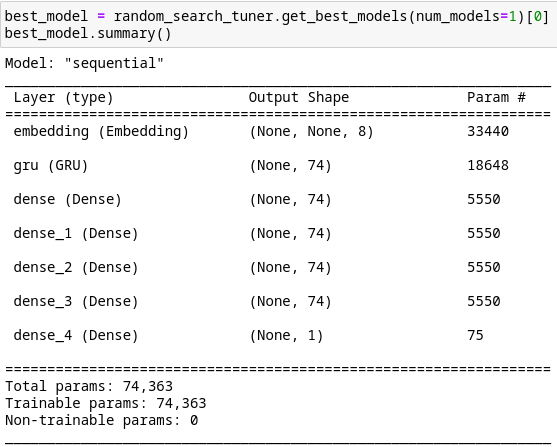
Now with the data cleaned, I can split the data into training, validation, and testing sets. The ratio I decided to use was a 70/15/15 train/val/test split, which is a common ratio used for sentiment analysis. Then I applied the tokenizer to the training split using the ‘fit\_on\_texts’ method, once fitted I then use the ‘texts\_to\_sequences’ method to convert the splits into sequences. The final step is to pad the sequences so they all have the same length, thankfully the Keras library provides the ‘pad\_sequences’ function allowing me to apply post-padding to each sequence. This results in every sequence having a length of 41, with padding tokens of ‘0’ being used to fill the empty sequence spaces.

To export the prepared dataset, I used the ‘to\_csv’ method from the Pandas library to export the final dataset into a csv file.

**B6:Prepared Dataset**

The prepared dataset will be attached to my submission titled ‘PA2\_cleaned\_data.csv’.

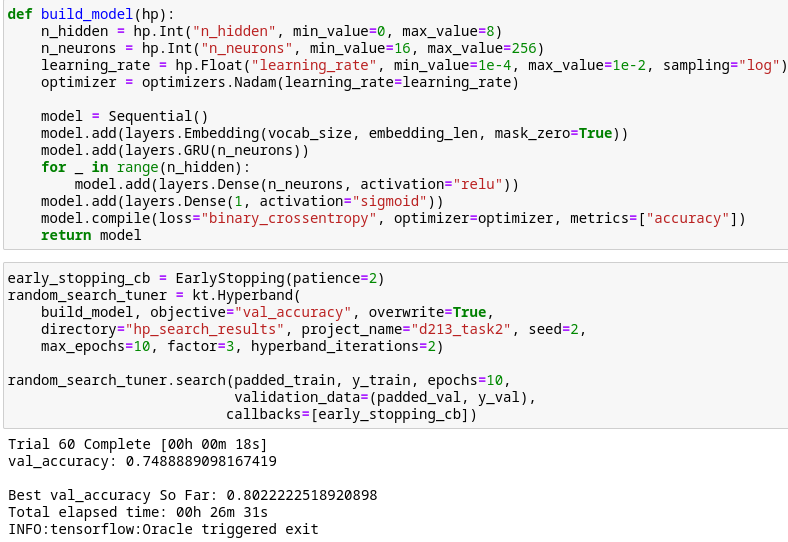
**C1:Model Summary**

****

**C2:Network Architecture**

The network architecture of my final model consists of an embedding input layer, a gated recurrent unit layer, 4 hidden dense layers, and a final dense output layer. An embedding layer is necessary for performing sentiment analysis to reduce the dimensions of the vectors representing each word. I chose a gated recurrent unit layer because GRU cells excel in processing sequential data. Dense layers are frequently used for classification problems such as sentiment analysis.

I used a hyperparameter tuner from the Keras library to decide on the optimal amount of hidden layers and parameters as well as other hyperparameters.

**C3:Hyperparameters**

I used a Sigmoid function for the final output layer because it works well for binary classification problems because of its binary nature. For the hidden dense layers, I chose the rectified linear activation function because it is a very common activation function for hidden layers and in my manual testing it outperformed other types of function.

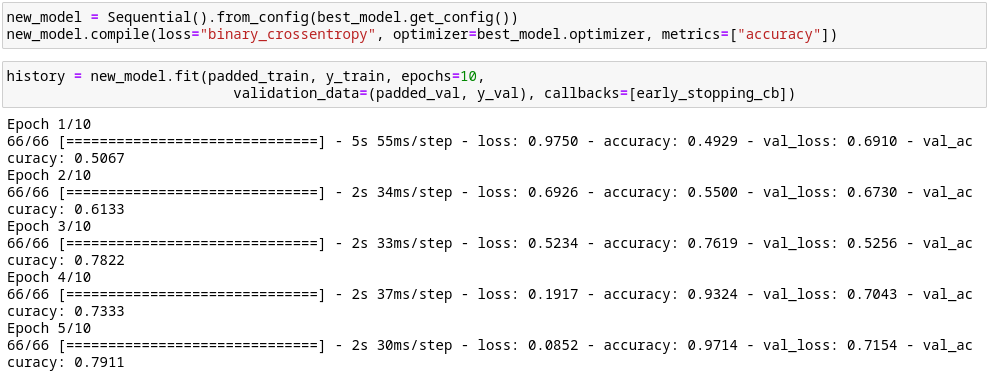
From a range of 16 to 256 nodes per layer, the hyperband tuner found that 74 resulted in the best validation accuracy score. Although it is common to have a different amount of neurons per hidden layer, I set it to stay constant for each layer because it performs just as well and in some cases even better, according to Géron (2022). This also comes with the benefit of having less hyperparameters to tune.

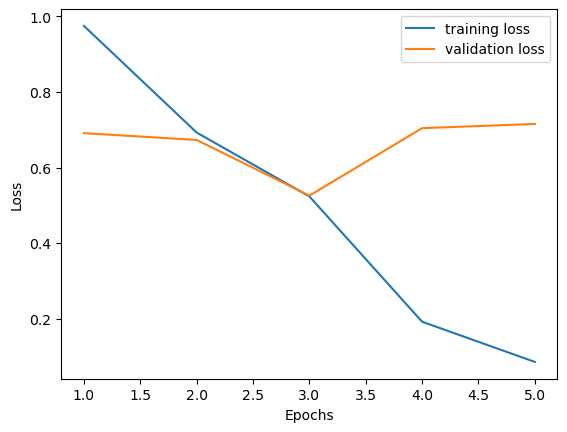
The loss function was set to a binary categorical crossentropy because it is designed for binary classification analysis which is perfect for this case. To speed up the training procces, I used the Nadam optimization algorithm, a variant of the popular Adam algorithm, this algorithm was found to generally outperform Adam in a study according to Géron (2022). From my manual testing it performed just as well and in some cases better than Adam, so I set it use Nadam by default to reduce the amount of hyperparameters to tune.

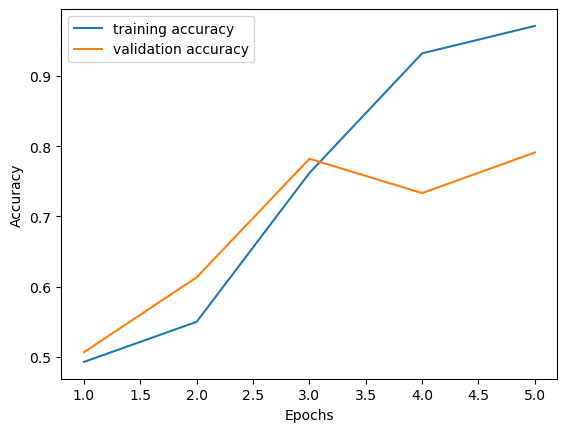
I used a stopping criteria to prevent the neural network from overfitting on the training data, I set it to stop after 2 epochs without improving the validation accuracy. My primary evaluation metric was classification accuracy, the model performs well in that aspect with a training accuracy of 98%, a validation accuracy of 80%, and a test accuracy of 81%.

**D1:Stopping Criteria**

The stopping criteria was crucial for preventing the neural network from overfitting on the training data and speeding up the hyperparameter tuning process by lowering the amount of epochs needed. For the screenshot of the final training epoch, unfortunately I could not retrieve the training process of my model because the hyperband tuner class from Keras does not save each epoch, only the resulting model and its metrics. So I simulated the training process with a new model using the exact same model parameters, the results are slightly different due to random chance.

**D2:Training Process**

****

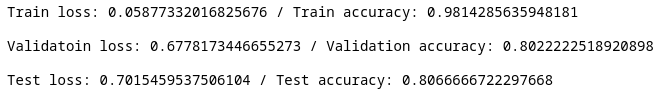
****

**D3:Fit**

The high training accuracy score of my model, 98%, suggests that it has overfitted the training data. The measures I used to attempt to prevent this from happening was using an early stopping criteria to prioritize validation accuracy improvement and I initially used a low amount of hidden layers and neurons for model testing to keep the neural network simple. I tested several models manually and found that simple neural networks performed just as well as complex ones, but the top models found by the ‘Hyperband’ parameter tuner from the Keras library outperformed the simple models in training, testing, and validation accuracy. The model could be improved by gathering more training data and simplifying the network.

**D4:Predictive Accuracy**

The trained network is very accurate at predicting the sentiment of the training set, at a 98% rate. The RNN also predicts the validation and testing sets’ sentiment correctly at an 80% rate. There is room for improvement, as the training set accuracy is much higher than the other sets, but the overall classification accuracy is quite good.

**E:Code**



**F:Functionality**

The functionality of the neural network I developed is that it can analyze the sentiment of reviews from three different sources, all discussing three different types of services or products, and give a reliable prediction as to whether that review was positive or negative. Its architecture makes a great impact on determining the sentiment of its input, each layer is deeply connected, parsing thousands of sentences and assigning weights to each vector, concluding with a prediction of 0 for negative or 1 for positive sentiment based off of the weights of each vector.

**G:Recommendations**

The neural network can adequately predict the sentiment of reviews from three different sources, all pertaining to different types of services. I would recommend this model for any organization with similar services that needs sentiment analysis as the model generalizes well and will only get better with more training data and improvements made to address overfitting.

**H:Reporting**

The report will be attached to my submission titled ‘PA2 Report.html’.

**I & J: Sources**

Géron, A. (2022). *Hands-on machine learning with scikit-learn, Keras, and tensorflow: Concepts, tools, and techniques to build Intelligent Systems* (3rd ed.). O'Reilly Media.

Kilinc, C. (2020, April 3). *Padding for NLP*. Medium. Retrieved March 14, 2023, from https://medium.com/@canerkilinc/padding-for-nlp-7dd8598c916a

Kapoor, A., Gulli, A., & Pal, S. (2022). *Deep Learning with Tensorflow and Keras*. Packt.

Kotzias, D., Denil, M., de Freitas, N., & Smyth, P. (2015). From group to individual labels using Deep Features. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://doi.org/10.1145/2783258.2783380

TensorFlow Team. (2017, November 20). *Introducing TensorFlow feature columns*. Google Developers Blog. Retrieved March 14, 2023, from https://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html