The Sentiment Analysis of Amazon Product Reviews: Natural Language Processing using a Deep Neural Network

Alexander Vaillant

Department of Information Technology, Western Governors University

D213: Advanced Data Analytics

Dr. Festus Elleh

July 30, 2021

Contents

Part One: Research Question

Research Question (A1)

To what extent can we accurately identify our future customer’s sentiment in unseen reviews based on the past reviews on our products by using NLP and NN techniques? The end model will be used as part of an early “warning” system to trigger the need for intervention and praise based on the review’s sentiment.

Objectives/Goals (A2)

1. A model with an accuracy of above 90% and a validation accuracy of above 75% on seen data.
2. A model with an accuracy of above 70% and loss under 0.60 on unseen data.
3. Since this model will be used for production, this model must be trained in under a minute.
    a. (This means that the model.fit() stage shouldn’t take longer than 1 minute.)

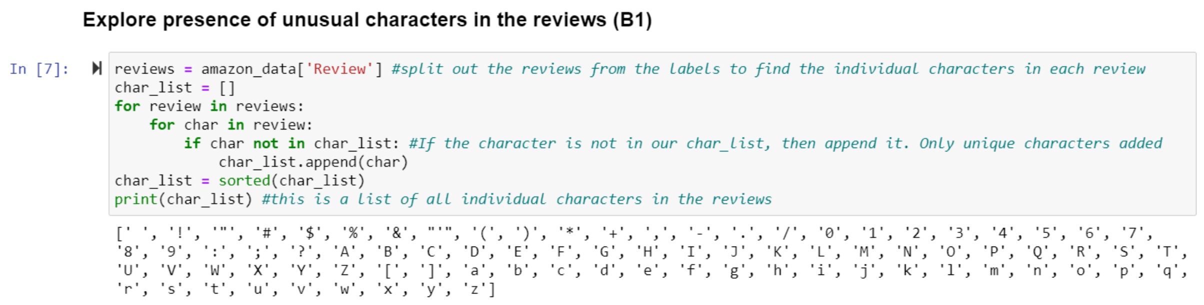
Type of Neural Network (A3)

I will be using a Deep Neural Network composed of an initial Embedding layer and multiple Dense layers.

Part Two: Data Preparation

Exploratory Data Analysis (B1)

1. Presence of Unusual Characters:

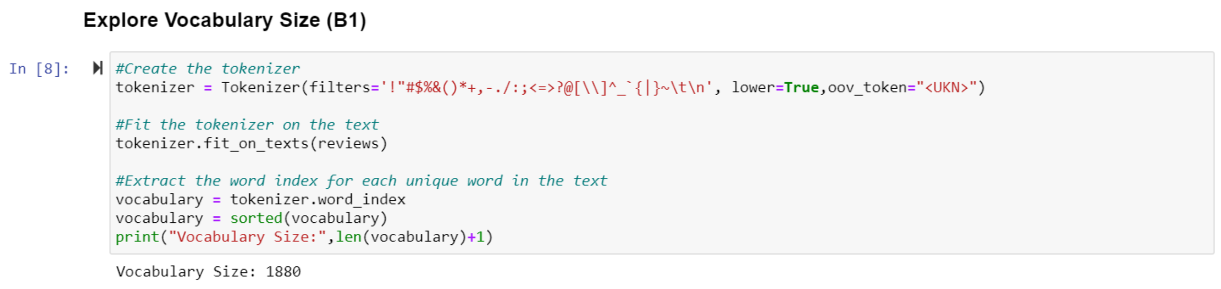


Explore presence of unusual characters in the reviews (B1)

```
In [7]:  reviews = amazon_data['Review'] #split out the reviews from the labels to find the individual characters in each review
         char_list = []
         for review in reviews:
             for char in review:
                 if char not in char_list: #If the character is not in our char_list, then append it. Only unique characters added
                     char_list.append(char)
         char_list = sorted(char_list)
         print(char_list) #this is a list of all individual characters in the reviews

         [' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7',
         '8', '9', ':', ';', '?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
         'U', 'V', 'W', 'X', 'Y', 'Z', '[', ']', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',
         'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

I set up a few for loops to parse each individual character from all reviews. If the character was not seen before, I appended the characters into the char_list. There were several punctuations and symbols found which needed to be parsed out of the data before the model can be fit with the training data. Also, the unique alphabet characters must be changed to lower cased.

2. Vocabulary Size:

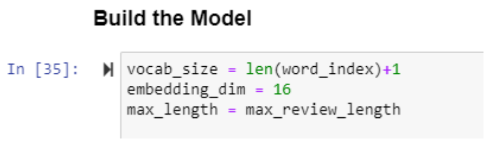


Explore Vocabulary Size (B1)

```
In [8]:  #Create the tokenizer
         tokenizer = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n', lower=True,oov_token="<UKN>")

         #Fit the tokenizer on the text
         tokenizer.fit_on_texts(reviews)

         #Extract the word index for each unique word in the text
         vocabulary = tokenizer.word_index
         vocabulary = sorted(vocabulary)
         print("Vocabulary Size:",len(vocabulary)+1)

         Vocabulary Size: 1880
```

In this analysis, the vocabulary size is the number of unique words in all our reviews. To get the entire vocabulary size of all reviews, including the test set, I used the Tokenizer from keras. I removed any unusual characters by using the filters parameter, lower-cased the reviews with the lower parameter and finally, fit the Tokenizer on the review data. I exported the word index, which is an index of each unique word, to the vocabulary variable. A length function was applied to vocabulary to get the final size vocabulary size of 1880.

3. **Proposed Word Embedding Length:**

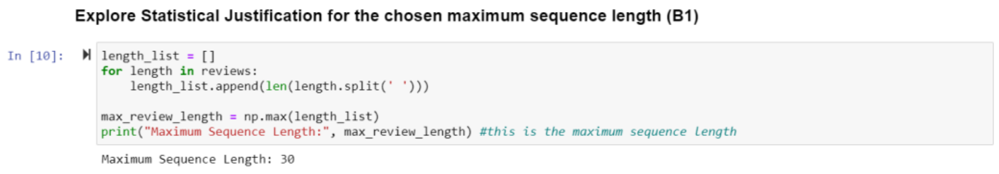


Build the Model

```
In [35]:  vocab_size = len(word_index)+1
          embedding_dim = 16
          max_length = max_review_length
```

The Word Embedding Length is the n-dimensions of a vector space in which the words will be embedded. For the proposed word embedding length, I used 16. Since 8 is the recommended minimum length proposed by the tensorflow team and it's based on the size of the vocabulary, I chose to use 16 because our vocabulary size is under 2,000 and we only have 1000 reviews.

4. **Statistical Justification for the Chosen Maximum Sequence Length:**



Explore Statistical Justification for the chosen maximum sequence length (B1)

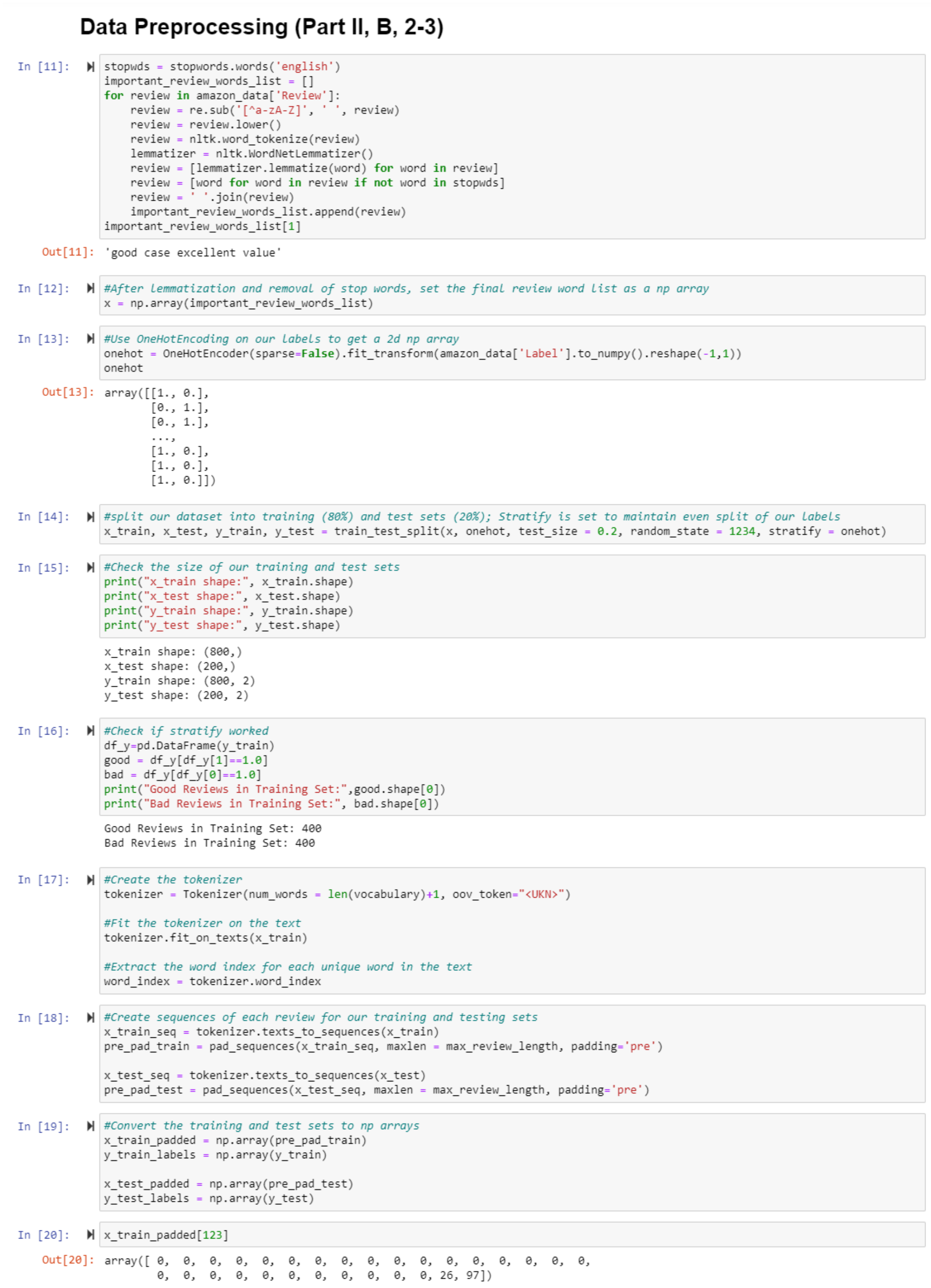
```
In [10]:  length_list = []
          for length in reviews:
              length_list.append(len(length.split(' ')))

          max_review_length = np.max(length_list)
          print("Maximum Sequence Length:", max_review_length) #this is the maximum sequence length

          Maximum Sequence Length: 30
```

For Statistical Justification of my Chosen Maximum Sequence Length, I used a for loop over my reviews dataframe and grabbed the length of each review. Once all the review lengths were found, I took the maximum length and established it as my max_review_length variable. If the maximum length of any of the 1000 reviews is only 30, then I should set my maximum sequence length to only 30 for the padding and model building phases.

Tokenization Process (B2)

1. **Packages used to normalize text:**
   a. Tensorflow.keras.preprocessing.text.Tokenizer ()
      i. To transform our reviews from text to embedded sequences and get the new vocabulary size of only the training data.
   b. Tensorflow.keras.preprocessing.sequence pad_sequences()
      i. To add padding to our embedded sequences.
   c. Re (Regular Expressions)
      i. To replace any non-letter characters and symbols with spaces.
   d. Nltk: word_tokenize, stopwords
      i. To remove stopwords from our reviews
   e. Nltk: wordnet
      i. To group together words with similar inflections (lemmatization)
2. **Goals of Tokenization Process:**
   1. Replace any unusual characters in our reviews
   2. Lower case all reviews
   3. Perform lemmatization to remove words with similar inflections
   4. Transform text reviews into sequences
   5. Pre-pad the sequences with the maximum sequence length of 30.

3. Tokenization Code



Data Preprocessing (Part II, B, 2-3)

```
In [11]:  stopwds = stopwords.words('english')
          important_review_words_list = []
          for review in amazon_data['Review']:
              review = re.sub('[^a-zA-Z]', ' ', review)
              review = review.lower()
              review = nltk.word_tokenize(review)
              lemmatizer = nltk.WordNetLemmatizer()
              review = [lemmatizer.lemmatize(word) for word in review]
              review = [word for word in review if not word in stopwds]
              review = ' '.join(review)
              important_review_words_list.append(review)
          important_review_words_list[1]

Out[11]:  'good case excellent value'
```

```
In [12]:  #After lemmatization and removal of stop words, set the final review word list as a np array
          x = np.array(important_review_words_list)
```

```
In [13]:  #Use OneHotEncoding on our labels to get a 2d np array
          onehot = OneHotEncoder(sparse=False).fit_transform(amazon_data['Label'].to_numpy().reshape(-1,1))
          onehot

Out[13]:  array([[1., 0.],
                 [0., 1.],
                 [0., 1.],
                 ...,
                 [1., 0.],
                 [1., 0.],
                 [1., 0.]])
```

```
In [14]:  #split our dataset into training (80%) and test sets (20%); Stratify is set to maintain even split of our labels
          x_train, x_test, y_train, y_test = train_test_split(x, onehot, test_size = 0.2, random_state = 1234, stratify = onehot)
```

```
In [15]:  #Check the size of our training and test sets
          print("x_train shape:", x_train.shape)
          print("x_test shape:", x_test.shape)
          print("y_train shape:", y_train.shape)
          print("y_test shape:", y_test.shape)

          x_train shape: (800,)
          x_test shape: (200,)
          y_train shape: (800, 2)
          y_test shape: (200, 2)
```

```
In [16]:  #Check if stratify worked
          df_y=pd.DataFrame(y_train)
          good = df_y[df_y[1]==1.0]
          bad = df_y[df_y[0]==1.0]
          print("Good Reviews in Training Set:",good.shape[0])
          print("Bad Reviews in Training Set:", bad.shape[0])

          Good Reviews in Training Set: 400
          Bad Reviews in Training Set: 400
```

```
In [17]:  #Create the tokenizer
          tokenizer = Tokenizer(num_words = len(vocabulary)+1, oov_token="<UKN>")

          #Fit the tokenizer on the text
          tokenizer.fit_on_texts(x_train)

          #Extract the word index for each unique word in the text
          word_index = tokenizer.word_index
```

```
In [18]:  #Create sequences of each review for our training and testing sets
          x_train_seq = tokenizer.texts_to_sequences(x_train)
          pre_pad_train = pad_sequences(x_train_seq, maxlen = max_review_length, padding='pre')

          x_test_seq = tokenizer.texts_to_sequences(x_test)
          pre_pad_test = pad_sequences(x_test_seq, maxlen = max_review_length, padding='pre')
```

```
In [19]:  #Convert the training and test sets to np arrays
          x_train_padded = np.array(pre_pad_train)
          y_train_labels = np.array(y_train)

          x_test_padded = np.array(pre_pad_test)
          y_test_labels = np.array(y_test)
```

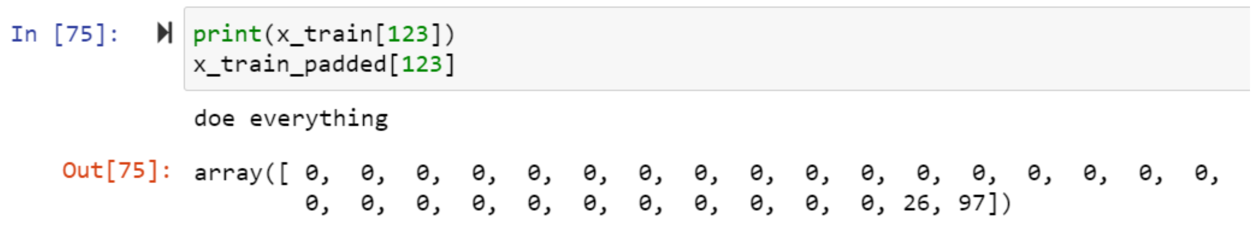
```
In [20]:  x_train_padded[123]

Out[20]:  array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
                  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 26, 97])
```

Padding Process (B3)

1. Explanation of Padding Process:

I chose to pre-pad my sequences. Since the maximum length of any review seen in our data was only 30, I chose to pre-pad my sequences to a maximum length of 30. This means that there would be a stream of zeros before the text's index number. The number of zeros would be equivalent to the maximum_review_length – the number of words in a review. In our example below, there are only two words and their index values are placed at the end of the sequence as I chose to pre-pad the sequences. If you count each number, then there are 30 numbers in total.

2. Screenshot of a single padded sequence:



```
In [75]:   print(x_train[123])
           x_train_padded[123]

           doe everything

Out[75]:   array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
                   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 26, 97])
```

Categories of Sentiment and Final Layer Activation Function (B4)

1. Categories of Sentiment:
    a. Embedding Layer (input layer)
    b. Flatten Layer
    c. Dense Layer (hidden layer)
    d. Dropout Layer
    e. Dense Layer (hidden layer)
    f. Dense Layer (output/final layer)
2. The final/output layer's activation function will be softmax.

Data Preparation Process Steps (B5)

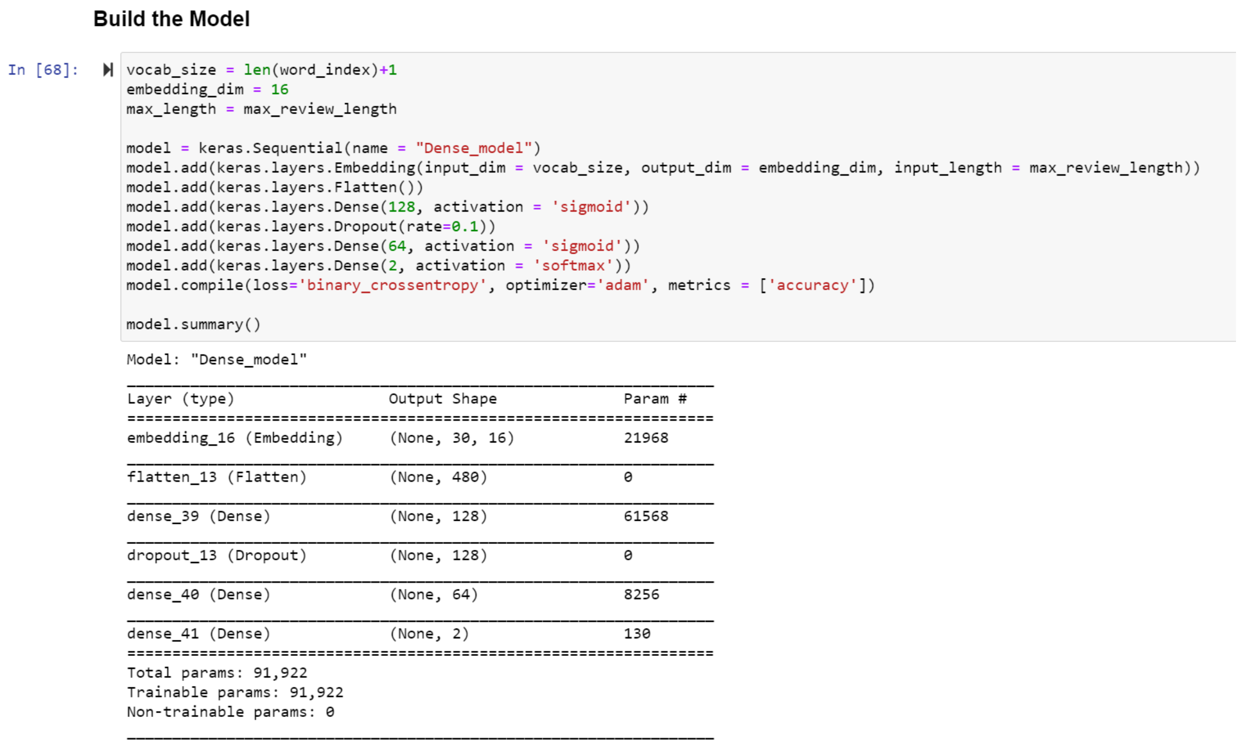
1. Load in the raw amazon review text data.
2. Perform EDA on the data:
    a. Explore the presence of unusual characters to be removed later.
    b. Find the entire dataset's vocabulary size to justify my proposed word embedding length.
    c. Find the maximum length of any review in our dataset to be used in padding the sequences.
3. Load in the stopwords to remove from our reviews and improve model accuracy.
4. Replace any non-alphabetic character with a space to improve model accuracy and clean up the reviews.
5. Lower case all reviews so there won't be any duplicate words due to lower vs. upper case.
6. Perform lemmatization to group words with similar inflections.
7. Remove any word that was in our stop words set from our reviews to help the neural network model learn.
8. OneHotEncode the labels into 2d numpy arrays.
9. Split the data into training and testing data sets. The training set has 800 reviews and the testing set has 200 reviews.
10. Get the final shape of each set.
11. Fit the tokenizer on the training set.
12. Get the vocabulary length of the training set which will be used when I build the model.
13. Transform the reviews of both the training and testing sets from text to sequences based on the word_index.
14. Pre-pad the sequences so each sequence has a maximum length of 30 which is equal the maximum length of any review in the entire dataset.
15. Convert the training and testing sets into np.arrays to fit the model on.
16. Export the final training and testing sets in .csv format.

Copy of Prepared Dataset (B6)

1. Please see "Training_Data.csv" for the x-values of the training data set (seen data).
2. Please see "Training_Labels.csv" for the y-values of the training data set (seen data).
3. Please see "Test_data.csv" for the x-values of the testing data set (unseen data)
4. Please see "Test_labels.csv" for the y-values of the testing data set (unseen data)

Part Three: Network Architecture

Output of Model Summary (C1)



Build the Model

```
In [68]:  vocab_size = len(word_index)+1
          embedding_dim = 16
          max_length = max_review_length

          model = keras.Sequential(name = "Dense_model")
          model.add(keras.layers.Embedding(input_dim = vocab_size, output_dim = embedding_dim, input_length = max_review_length))
          model.add(keras.layers.Flatten())
          model.add(keras.layers.Dense(128, activation = 'sigmoid'))
          model.add(keras.layers.Dropout(rate=0.1))
          model.add(keras.layers.Dense(64, activation = 'sigmoid'))
          model.add(keras.layers.Dense(2, activation = 'softmax'))
          model.compile(loss='binary_crossentropy', optimizer='adam', metrics = ['accuracy'])

          model.summary()
```

```
Model: "Dense_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_16 (Embedding)     (None, 30, 16)            21968
_________________________________________________________________
flatten_13 (Flatten)         (None, 480)               0
_________________________________________________________________
dense_39 (Dense)             (None, 128)               61568
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_40 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_41 (Dense)             (None, 2)                 130
=================================================================
Total params: 91,922
Trainable params: 91,922
Non-trainable params: 0
_________________________________________________________________
```

Discuss layers and parameters (C2)

Num. of Layer	Type of Layer	Macro Type of Layer	Num. of Parameters
1	Embedding	Input	21,968
2	Flatten	Flatten	0
3	Dense	Hidden	61,568
4	Dropout	Dropout	0
5	Dense	Hidden	8,256
6	Dense	Output	130

There are a total of 6 layers and 91,922 parameters (also trainable). I started with an Embedding input layer and flattened the output shape from (,30,16) to (, 480). A dense, or “fully connected”, layer was added with 128 units. This is followed by a dropout layer with a rate of 0.1 to remove noise from the model and avoid overfitting. Another dense layer is introduced, but with 64 units. Finally, the Dense output layer is added with only 2 units and an activation function of softmax.

Justify hyperparameters (C3)

1. Activation Functions = (sigmoid & softmax)

I chose to use two different activation functions: sigmoid and softmax. The sigmoid activation function was used in the two dense, hidden layers. Sigmoid is great non-linear activation function for binary classification problems because it exists between 0 and 1, which is used to predict the probability of a binary class output (Sharma, 2017). If there were more than two classes, then I would have used a ‘relu’ activation function. The softmax activation function was used in the last dense, output layer. Typically, softmax is used in multi-class classification problems (Brownlee, 2020). However, it can still be used in binary classification problems. In this analysis, the softmax function produces the binary probability distribution of a review. I could have also used the sigmoid activation function again for the output layer, but the results produced were much lower in both training and validation accuracy.

2. Number of Nodes per Layer

For the input layer, I used the proposed word embedding length of 16 as my output dimension. I set the input_dimension as the training set’s vocabulary size. For my two dense, hidden layers, I specified the number of nodes, or “units”, as 128 and 64, respectively. I wanted the model to output 128 nodes and 64 nodes into the next respective layer. I did not want there to be too many output nodes which would exponentially increase the number of trainable parameters. This would extend the time it would take for the model to be trained. Since one of my objectives was to keep the model training period under one minute, it was important to limit the number of nodes for speed and efficiency’s sake. For my final dense, output layer, I specified the number of nodes as 2, which would output the final probability distribution of our two classes given the softmax activation function.

3. Loss Function = “binary_crossentropy”

I chose to use the loss function of “binary_crossentropy” while compiling my model. First, this is a classification, not regression analysis. I knew I had to choose a classification loss function. Binary_crossentropy was chosen because there are only two (binary) potential classes that a review can be categorized as in this analysis. These are “positive” and “negative”. If I had multiple potential classes, then I would have chosen “categorical_crossentropy”.

4. Optimizer = ‘Adam’ Algorithm

I chose to use the Adam Algorithm as my optimizer because it updates the neural network’s weights iteratively based on the training data (Brownlee, 2021). This helps reduce overfitting and improve model accuracy. Also, it’s slightly less complicated to use as compared to optimizers, like SGD, or Stochastic Gradient Descent, while still providing decent results.

5. Stopping Criteria = EarlyStopping(monitor = ‘val_accuracy’, patience = 3)

I chose to use an EarlyStopping callback. The stopping criteria were a monitor on validation accuracy score and a patience of 3. This would stop the model from training early if the validation accuracy didn’t increase overall after 3 epochs. In my model, the stopping criteria was not violated, and all 15 epochs were allowed to run.

6. Evaluation Metric = ‘Accuracy’

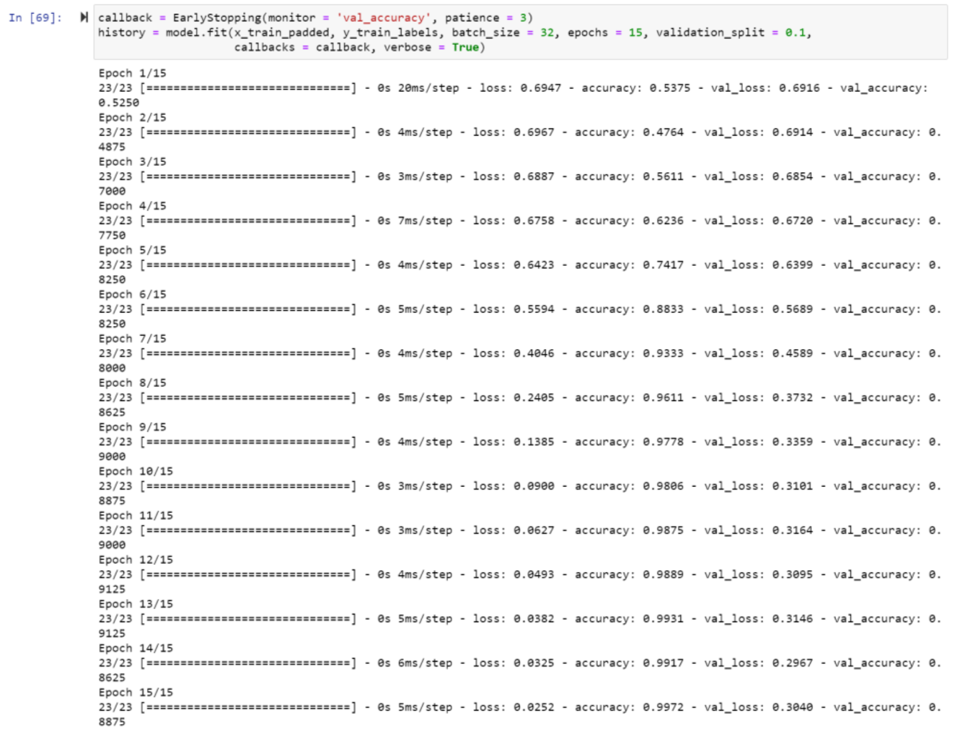
I chose to use the evaluation metric of “accuracy” for part of my model’s compiling phase. This was to evaluate the model’s overall ability to accurately predict a review’s sentiment class in the seen/training. Accuracy can be a range of 0 – 100%. Since I added a validation split, the model would also test its accuracy on the validation data. This would help indicate if the model was being overfitted.

Part Four: Model Evaluation

Impact of Using Stopping Criteria (D1)

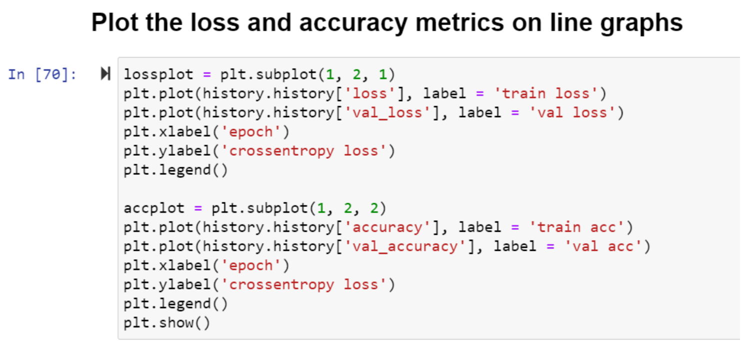
Unfortunately, it is impossible to train a model without defining the number of epochs as requested in this assessment. **If you tried to do so, the model would only run for one epoch.** However, you can specify both the number of epochs and an early stopping monitor at the same time. This is the approach I took. I used the EarlyStopping() function from keras and set the monitor to validation accuracy and patience to 3. This means that if the model's validation accuracy score does not improve after 3 consecutive epochs, then the model will stop training to avoid overfitting. Setting an early stopping monitor and patience is a great way to reduce potential overfitting of a neural network model. If the validation accuracy was to continue 'plunging' lower than an acceptable range in relation to the training accuracy, then this would be a case of overfitting. **In section "Visualizations of Model's Training Process (D2), you can see (below) that the model reached the final epoch of 15 without violating the stopping criteria I set.**

Visualizations of Model's training process (D2)



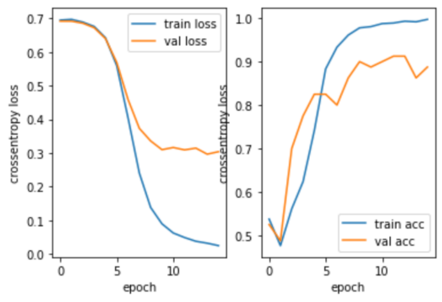
```
In [69]:  callback = EarlyStopping(monitor = 'val_accuracy', patience = 3)
          history = model.fit(x_train_padded, y_train_labels, batch_size = 32, epochs = 15, validation_split = 0.1,
                              callbacks = callback, verbose = True)

          Epoch 1/15
          23/23 [==============================] - 0s 20ms/step - loss: 0.6947 - accuracy: 0.5375 - val_loss: 0.6916 - val_accuracy:
          0.5250
          Epoch 2/15
          23/23 [==============================] - 0s 4ms/step - loss: 0.6967 - accuracy: 0.4764 - val_loss: 0.6914 - val_accuracy: 0.
          4875
          Epoch 3/15
          23/23 [==============================] - 0s 3ms/step - loss: 0.6887 - accuracy: 0.5611 - val_loss: 0.6854 - val_accuracy: 0.
          7000
          Epoch 4/15
          23/23 [==============================] - 0s 7ms/step - loss: 0.6758 - accuracy: 0.6236 - val_loss: 0.6720 - val_accuracy: 0.
          7750
          Epoch 5/15
          23/23 [==============================] - 0s 4ms/step - loss: 0.6423 - accuracy: 0.7417 - val_loss: 0.6399 - val_accuracy: 0.
          8250
          Epoch 6/15
          23/23 [==============================] - 0s 5ms/step - loss: 0.5594 - accuracy: 0.8833 - val_loss: 0.5689 - val_accuracy: 0.
          8250
          Epoch 7/15
          23/23 [==============================] - 0s 4ms/step - loss: 0.4046 - accuracy: 0.9333 - val_loss: 0.4589 - val_accuracy: 0.
          8000
          Epoch 8/15
          23/23 [==============================] - 0s 5ms/step - loss: 0.2405 - accuracy: 0.9611 - val_loss: 0.3732 - val_accuracy: 0.
          8625
          Epoch 9/15
          23/23 [==============================] - 0s 4ms/step - loss: 0.1385 - accuracy: 0.9778 - val_loss: 0.3359 - val_accuracy: 0.
          9000
          Epoch 10/15
          23/23 [==============================] - 0s 3ms/step - loss: 0.0900 - accuracy: 0.9806 - val_loss: 0.3101 - val_accuracy: 0.
          8875
          Epoch 11/15
          23/23 [==============================] - 0s 3ms/step - loss: 0.0627 - accuracy: 0.9875 - val_loss: 0.3164 - val_accuracy: 0.
          9000
          Epoch 12/15
          23/23 [==============================] - 0s 4ms/step - loss: 0.0493 - accuracy: 0.9889 - val_loss: 0.3095 - val_accuracy: 0.
          9125
          Epoch 13/15
          23/23 [==============================] - 0s 5ms/step - loss: 0.0382 - accuracy: 0.9931 - val_loss: 0.3146 - val_accuracy: 0.
          9125
          Epoch 14/15
          23/23 [==============================] - 0s 6ms/step - loss: 0.0325 - accuracy: 0.9917 - val_loss: 0.2967 - val_accuracy: 0.
          8625
          Epoch 15/15
          23/23 [==============================] - 0s 5ms/step - loss: 0.0252 - accuracy: 0.9972 - val_loss: 0.3040 - val_accuracy: 0.
          8875
```

Please see above for the visualization of each epoch's metric output. Please note that even though there was a EarlyStopping Monitor set for validation accuracy and a patience level of 3, the model was able to run each of the 15 epochs because the EarlyStopping Monitor was not violated.



Plot the loss and accuracy metrics on line graphs

In [70]:
```
lossplot = plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label = 'train loss')
plt.plot(history.history['val_loss'], label = 'val loss')
plt.xlabel('epoch')
plt.ylabel('crossentropy loss')
plt.legend()

accplot = plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label = 'train acc')
plt.plot(history.history['val_accuracy'], label = 'val acc')
plt.xlabel('epoch')
plt.ylabel('crossentropy loss')
plt.legend()
plt.show()
```

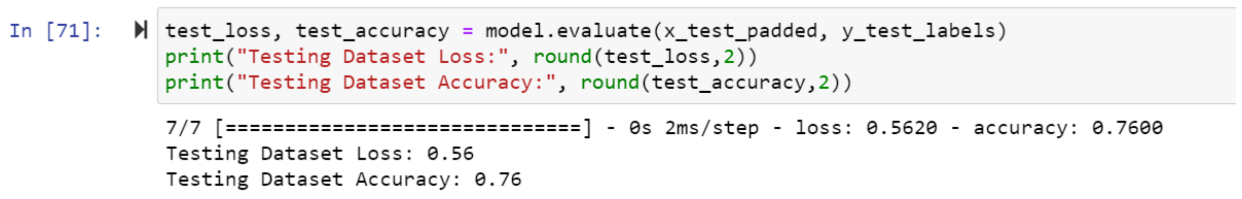


Please see above for the visualization of each epoch's metric output in line graph form. The line graph of the model's training and validation loss is on the left. The line graph of the model's training and validation accuracy is on the right.

Fitness of Model & Overfitting Avoidance Methods (D3)

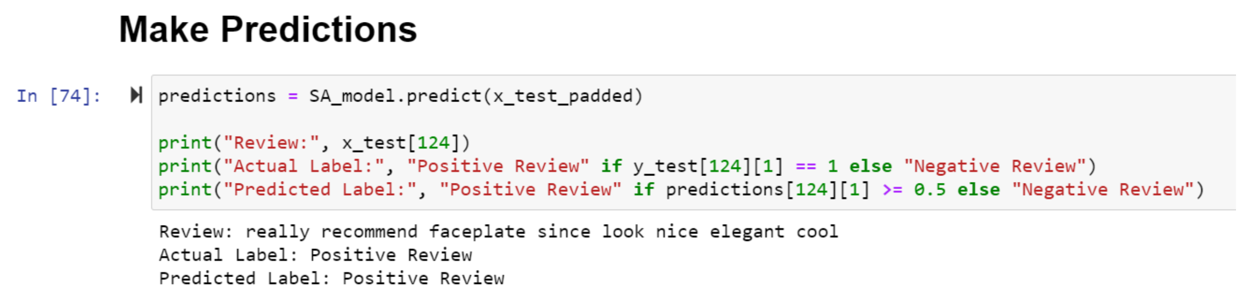
1. Fitness of Model:
   a. The trained model ended with a training accuracy and loss of 99.72% and 0.0252, respectively.
   b. The trained model ended with a validation accuracy and loss of 88.75% and 0.3040, respectively.
   c. Based on the graphs, you can see some leveling out of the validation loss towards the last half of epochs. Also, please note that there is some jagged-ness to the validation accuracy. These are some slight indications of overfitting, but nothing too serious.
2. Overfitting Avoidance Method:
   a. I used a Dropout layer with a rate of 0.1 in my model to remove any noise that can cause overfitting.
   b. I added an early stopping monitor with two parameters of monitor = validation accuracy and patience = 3. This would stop the model from training if the validation accuracy didn't increase overall after 3 epochs.
   c. I didn't train on the entire dataset and split the dataset into training and test sets.
   d. I used the Adam optimizer algorithm which updates the network's weights iteratively based on the training data (Brownlee, 2021). This helps reduce overfitting.
   e. Another method I could have tried would have been the shuffle=True parameter when fitting the model with the training data.

Predictive Accuracy of Model (D4)



```
In [71]:  test_loss, test_accuracy = model.evaluate(x_test_padded, y_test_labels)
          print("Testing Dataset Loss:", round(test_loss,2))
          print("Testing Dataset Accuracy:", round(test_accuracy,2))

          7/7 [==============================] - 0s 2ms/step - loss: 0.5620 - accuracy: 0.7600
          Testing Dataset Loss: 0.56
          Testing Dataset Accuracy: 0.76
```

I tested the trained model’s accuracy on the unseen data in the test datasets. When using model.evaluate() to evaluate the accuracy and loss of the model, the resulting predictive loss and accuracy was 0.56 and 76%. Both values of 0.56 and 76% satisfy the original objectives of under 0.60 and above 70% which I set for the model.



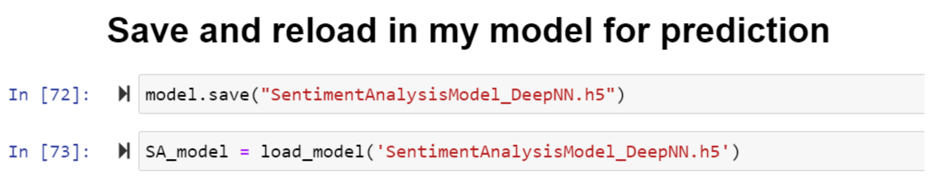
Make Predictions

```
In [74]:  predictions = SA_model.predict(x_test_padded)

          print("Review:", x_test[124])
          print("Actual Label:", "Positive Review" if y_test[124][1] == 1 else "Negative Review")
          print("Predicted Label:", "Positive Review" if predictions[124][1] >= 0.5 else "Negative Review")

          Review: really recommend faceplate since look nice elegant cool
          Actual Label: Positive Review
          Predicted Label: Positive Review
```

Also, I tested the model’s prediction accuracy further on an individual review. The actual label of the review was a positive review and the model also predicted that it was a positive review.

Part Five: Summary & Recommendations

Code to Save the NN model (E)



Save and reload in my model for prediction

```
In [72]:  model.save("SentimentAnalysisModel_DeepNN.h5")

In [73]:  SA_model = load_model('SentimentAnalysisModel_DeepNN.h5')
```

The model is saved as "SentimentAnalysisModel_DeepNN.h5" in .h5 format by using the code model.save(). The model is then re-loaded into our environment by using the load_model() function from keras.models. Once the model is saved, it can be loaded into any future environments of my choosing and used on cleansed data like the prepared datasets.

Functionality and Impact (F)

1. Functionality of Neural Network

The functionality of my neural network is to process cleansed review text data and classify/predict the sentiment of the review as either positive or negative. The neural network will also output the predictive probability of each class. The neural network can be used in production at a company to identify reviews that may require triage and added support for negatively impacted customers. Before it can be used in production, it must clear certain accuracy metric requirements which have been established in the above explanations.

2. Impact of Network Architecture

There are six total layers in my neural network. It starts first with an embedding layer to further group words of similar direction. The output of the embedding layer is fed into a flatten layer which re-shapes the dimensions of the embedding layer's output. The output of the flatten layer is fed into a dense, fully connected layer for the model to gleam insights. A dropout layer is introduced to reduce noise that may cause overfitting in our model. The last dense hidden layer is introduced for increased learning, but with much smaller units. Lastly, the final dense output layer of the model outputs an array with predictive probabilities for each of the two classes. This model accounts for overfitting with a dropout layer and early stopping criteria. The resulting model accurately predicts sentiment on unseen data above the 70% accuracy threshold.

Recommendation (G)

Given that the model is performing on unseen data with an accuracy of over 75%, I would recommend moving forward with this model and implementing it as a warning system that a bad review has been posted. Company representatives can be alerted that a bad review was posted, intervene by reaching out to the specific customer who posted the bad review, and hopefully reduce customer churn by providing solutions and excellent customer service. To test and increase its effectiveness, I would introduce the model to larger amounts of product reviews over the next month. After all, the model was only trained, tested, and validated on only 1000 product reviews.

Part Six: Reporting

Jupyter Notebook (H)

Please see "Task 2 Tensorflow NLP.HTML" and "Task 2 Tensorflow NLP.ipynb" for the HTML version of my Jupyter Notebook and the Jupyter Notebook itself. **Also, please note that model training and outputs can change on every run of the code.**

Web Sources (I & J)

Brownlee, J. (2021, January 13). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

Brownlee, J. (2020, October 19). *Softmax Activation Function with Python*. Machine Learning Mastery. https://machinelearningmastery.com/softmax-activation-function-with-python/.

Sharma, S. (2017, September 6). *Activation Functions in Neural Networks*. Towards Data Science. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

Professionalism (K)

Professional communication can be reviewed throughout this document and my jupyter notebook presentation.