My research question for this analysis is "Utilizing sentiment analysis with neural networks for analyzing labeled review sentiments can the sentiments be identified as positive or negative?"

The main objective of this analysis is to achieve a classification review of text with a positive or negative sentiment.

This analysis is accomplished with Tensorflow and Keras as well as nltk and sklearn.

In order to analyze the data first critical libraries and packages needed to be installed. Specifically, Tensorflow and Keras as well as nltk and sklearn. Next, three csv files which had been previously edited with excel to convert from txt files to csv and are titled "imdb", "yelp", and "amazon" were imported into the Pandas data frame. They were assigned a source and concated with Pandas. Next, the head of the data was evaluated as well as the data types. The index was reset for better analysis and the data was evaluated for null values. In addition, the dataset was reviewed to confirm the rating system was in binary format of 0 and 1. That was confirmed. The merged files were reviewed for positive and negative reviews. There were 1386 results of 1 and 1362 of 0 which was an even split. The data was analyzed for the string length and the characters in the "review" column were evaluated and non-alpha characters were removed before reviewing to confirm.

In [61]:
```
#imports for sentiment analysis with neural networks
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import seaborn as sns

import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR')


import tensorflow
from tensorflow import keras
import tensorflow.compat.v1 as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.callbacks import EarlyStopping


from dateutil.parser import parse

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer


from sklearn.model_selection import train_test_split


import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
#read csv with headers into Pandas dataframe
imdb_df = pd.read_csv('imdb.csv')
yelp_df = pd.read_csv('yelp.csv')
amazon_df = pd.read_csv('amazon.csv')
```

In [5]:
```
#assign a source
amazon_df['source']= 'amazon'
yelp_df['source']= 'yelp'
imdb_df['source']= 'imdb'
```

In [6]:
```
#combine files
df = pd.concat([amazon_df, imdb_df, yelp_df])
```

In [7]:
```
# data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2748 entries, 0 to 999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   review  2748 non-null   object
 1   score   2748 non-null   int64 
 2   source  2748 non-null   object
dtypes: int64(1), object(2)
memory usage: 85.9+ KB
```

In [8]:
```
#first look at top of data
df.head()
```

Out[8]:

	review	score	source
0	So there is no way for me to plug it in here i...	0	amazon
1	Good case, Excellent value.	1	amazon
2	Great for the jawbone.	1	amazon
3	Tied to charger for conversations lasting more...	0	amazon
4	The mic is great.	1	amazon

In [9]:
```
df=df.reset_index(drop=True)
```

In [10]:
```
#discover missing data points within dataset
data_nulls = df.isnull().sum()
print(data_nulls)
```

```
review    0
score     0
source    0
dtype: int64
```

In [11]:
```
#confirm binary rating system of 0 = negative, 1 = positive
df.score.unique()
```

Out[11]:
```
array([0, 1], dtype=int64)
```

In [12]:
```
#review positive and negative reviews from merged DataFrame
print('Merged Dataframe \n', df.score.value_counts())
```

```
Merged Dataframe
 1    1386
0    1362
Name: score, dtype: int64
```

In [13]:
```
#find review lengths
length_review = df.review.str.len()
length_review
```

Out[13]:
```
0        82
1        27
2        22
3        79
4        17
       ...
2743     66
2744     24
2745     50
2746     91
2747    134
Name: review, Length: 2748, dtype: int64
```

In [14]:
```
#print reviews type
type(length_review)
```

Out[14]:
```
pandas.core.series.Series
```

In [15]:
```
#find shortest review
min(length_review)
```

Out[15]:
```
6
```

In [16]:
```
#sum all reviews
sum(length_review)
```

Out[16]:
```
196492
```

In [17]:
```
#print length of the longest review
print('Longest review is ' + str(max(length_review)) + ' words.')
```

```
Longest review is 7944 words.
```

In [18]:
```
#perform bag of words tokenization
#import bag count vectorizer function from Scikit-Learn
from sklearn.feature_extraction.text import CountVectorizer

#limit features to the top 20 words
vect = CountVectorizer(max_features=20)

#fit count vectorizer
vect.fit(df.review)
```

Out[18]:
```
CountVectorizer(max_features=20)
```

In [19]:
```
#identify characters that appear in the dataset
characters = {}
```

```
for s in df['review']:
    for l in s:
        if l in characters.keys():
            characters[l] += 1
        else:
            characters[l] = 1
print(len(characters))
sorted(["{}: {}".format(k, v) for k,v in characters.items()])
```

```
89
```

Out[19]:
```
['\t: 252',
 '\n: 252',
 ' : 34473',
 '!: 501',
 '": 120',
 '#: 1',
 '$: 17',
 '%: 5',
 '&: 27',
 "': 723",
 '(: 89',
 '): 89',
 '*: 16',
 '+: 6',
 ',: 1305',
 '-: 292',
 '.: 3082',
 '/: 40',
 '0: 270',
 '1: 219',
 '2: 77',
 '3: 46',
 '4: 28',
 '5: 64',
 '6: 8',
 '7: 24',
 '8: 28',
 '9: 29',
 ':: 37',
 ';: 24',
 '?: 28',
 'A: 382',
 'B: 248',
 'C: 175',
 'D: 175',
 'E: 296',
 'F: 122',
 'G: 172',
 'H: 185',
 'I: 1383',
 'J: 69',
 'K: 35',
 'L: 159',
 'M: 173',
 'N: 218',
 'O: 200',
 'P: 184',
 'Q: 16',
 'R: 164',
 'S: 313',
 'T: 978',
 'U: 63',
 'V: 138',
 'W: 236',
 'X: 8',
 'Y: 83',
 'Z: 11',
 'a: 12137',
 'b: 2301',
 'c: 4372',
```

```
'd: 5821',
'e: 18631',
'f: 3136',
'g: 3118',
'h: 7534',
'i: 10425',
'j: 220',
'k: 1326',
'l: 6686',
'm: 3557',
'n: 9378',
'o: 11347',
'p: 2876',
'q: 144',
'r: 8681',
's: 9739',
't: 13601',
'u: 4095',
'v: 2028',
'w: 3115',
'x: 286',
'y: 3399',
'z: 164',
'\x85: 2',
'\x96: 5',
'\x97: 1',
'å: 1',
'é: 7',
'ê: 1']
```

In [20]:
```
#remove any non-alpha character
df["review"] = df["review"].str.replace("[^a-zA-Z ]", "", regex=True)
```

In [21]:
```
#check character counts
characters = {}
for s in df['review']:
    for l in s:
        if l in characters.keys():
            characters[l] += 1
        else:
            characters[l] = 1
print(len(characters))
sorted(["{}: {}".format(k, v) for k,v in characters.items()])
```

```
53
```

Out[21]:
```
[' : 34473',
 'A: 382',
 'B: 248',
 'C: 175',
 'D: 175',
 'E: 296',
 'F: 122',
 'G: 172',
 'H: 185',
 'I: 1383',
 'J: 69',
 'K: 35',
 'L: 159',
 'M: 173',
 'N: 218',
 'O: 200',
 'P: 184',
 'Q: 16',
 'R: 164',
 'S: 313',
 'T: 978',
 'U: 63',
 'V: 138',
 'W: 236',
 'X: 8',
 'Y: 83',
 'Z: 11',
 'a: 12137',
 'b: 2301',
 'c: 4372',
 'd: 5821',
 'e: 18631',
 'f: 3136',
 'g: 3118',
 'h: 7534',
 'i: 10425',
 'j: 220',
 'k: 1326',
 'l: 6686',
 'm: 3557',
 'n: 9378',
 'o: 11347',
 'p: 2876',
 'q: 144',
 'r: 8681',
 's: 9739',
 't: 13601',
 'u: 4095',
 'v: 2028',
 'w: 3115',
 'x: 286',
 'y: 3399',
 'z: 164']
```

In [22]:
```
#define word counting function
def word_cnt(arr):
    words = {}
    for s in arr:
        for w in s:
            w = w.lower()
```

```
            if w in words.keys():
                words[w] += 1
            else:
                words[w] = 1
    return words
```

Unique word count was performed and found to be 5,302. Next, sequences were examined.

In [23]:
```
#unique word count
len(word_cnt([w for w in [s.split(' ') for s in df['review']]]))
```

Out[23]:
```
5303
```

In [24]:
```
#examine sequences
sentence_len = np.array([len(s.split(' ')) for s in df['review']])
print("Longest sequence: {}".format(sentence_len.max()))
print("Shortest sequence: {}".format(sentence_len.min()))
print("Mean sequence: {}".format(sentence_len.mean()))
print("Median sequence: {}".format(np.median(sentence_len)))
print("Total words: {}".format(np.sum(sentence_len)))
```

```
Longest sequence: 1393
Shortest sequence: 1
Mean sequence: 13.54475982532751
Median sequence: 11.0
Total words: 37221
```

Utilizing Lemmatization with WordNetLemmatizer any sentences with empty records were removed and the text was converted to all lowercase and converted to list format. After this grouping the data was analyzed for the unique words after the reductions and showed a new total unique word count of 4,764.

In [25]:
```
#remove any records with empty sentences
df = df[df['review'].str.len() != 0]
```

In [26]:
```
#lower case, remove stop words, covert each sentence to list
df['review'] = df['review'].apply(
    lambda s: [word.lower() for word in s.split() if word.lower() not in stopwords.wor
)
```

In [27]:
```
#lemmatization
lemmatizer = WordNetLemmatizer()
df['review'] = df['review'].apply(lambda s: [lemmatizer.lemmatize(w) for w in s])
```

In [29]:
```
#unique words after reductions
all_words = word_cnt(df['review'])
print("Total unique words: {}".format(len(all_words)))
```

```
Total unique words: 4765
```

Rare words were removed and the unique word count after that removal is 1988

In [31]:
```
# # of words that only appear once
rare_words = [k for k,v in all_words.items() if v == 1]
len(rare_words)
```

Out[31]:
```
2777
```

In [32]:
```
#remove rare words
df['review'] = df['review'].apply(
    lambda s: [word for word in s if word not in rare_words]
)
#remove any records with empty sentences
df = df[df['review'].str.len() != 0]
```

In [33]:
```
#unique words after removing rare words
print("Total unique words: {}".format(len(word_cnt(df['review']))))
```

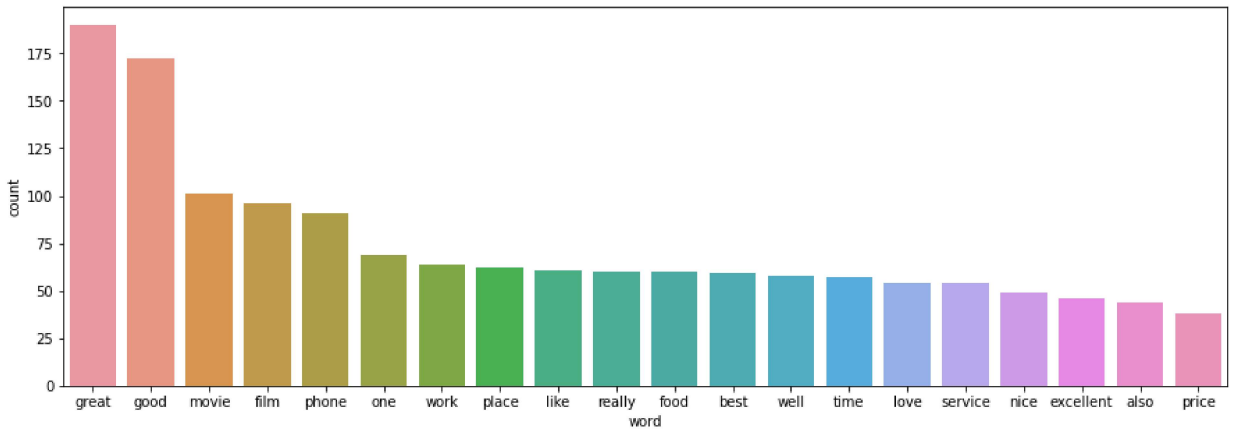
```
Total unique words: 1988
```

In [34]:
```
#sequence stats after reductions
sentence_len = np.array([len(s) for s in df['review']])
print("Longest sequence: {}".format(sentence_len.max()))
print("Shortest sequence: {}".format(sentence_len.min()))
print("Mean sequence: {}".format(sentence_len.mean()))
print("Median sequence: {}".format(np.median(sentence_len)))
print("Total words: {}".format(np.sum(sentence_len)))
```

```
Longest sequence: 554
Shortest sequence: 1
Mean sequence: 5.698099415204679
Median sequence: 4.0
Total words: 15590
```

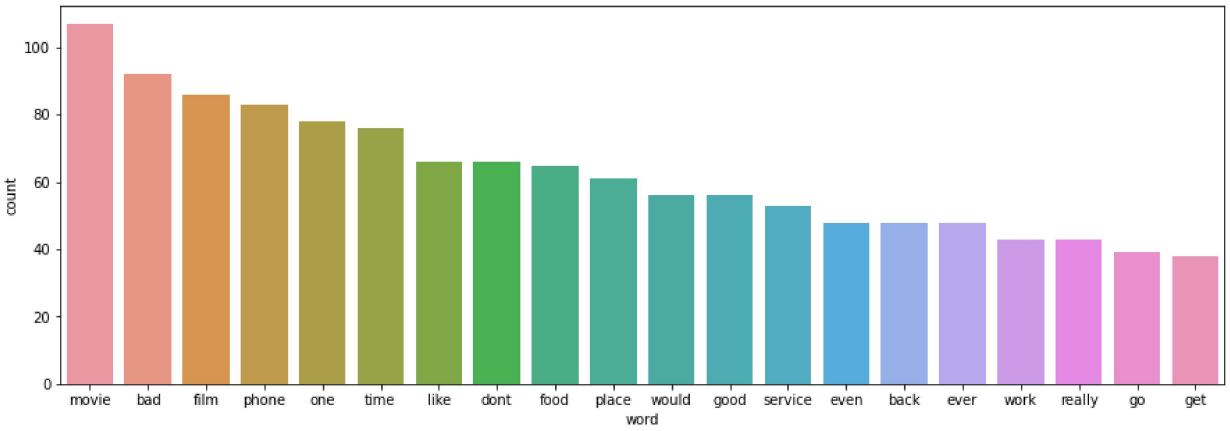
In [35]:
```
##isolate positive and negative sequences
pos = df[df['score'] == 1]
pos = pd.DataFrame(sorted(word_cnt(pos['review']).items(), key=lambda item: item[1], r
neg = df[df['score'] == 0]
neg = pd.DataFrame(sorted(word_cnt(neg['review']).items(), key=lambda item: item[1], r
```

In [36]:
```
#plot top 20 words in positive sentiment reviews
plt.figure(figsize=(15, 5))
sns.barplot(x = 'word',
            y = 'count',
            data = pos.head(20))

#show the plot
plt.show()
```



In [37]:
```
#plot top 20 words in negative sentiment reviews
```

```
plt.figure(figsize=(15, 5))
sns.barplot(x = 'word',
            y = 'count',
            data = neg.head(20))

#show the plot
plt.show()
```



In [38]:
```
#look at words that appear frequently in both sentiments
common = pos[pos['word'].isin(neg['word'])]
common.head(10)
```

Out[38]:

	word	count
0	great	190
1	good	172
2	movie	101
3	film	96
4	phone	91
5	one	69
6	work	64
7	place	62
8	like	61
9	really	60

In [40]:
```
#define neutral words that appear frequently in both sets
neutral = ['film','movie','phone','one','place']
# remove neutral words
df['review'] = df['review'].apply(
    lambda s: [word for word in s if word not in neutral]
)
#remove any records with empty sentences
df = df[df['review'].str.len() != 0]
```

In [41]:
```
#unique words after removing common words
print("Total unique words: {}".format(len(word_cnt(df['review']))))
```

```
Total unique words: 1983
```

In [42]:
```
#sequence stats
review_len = np.array([len(s) for s in df['review']])
print("Longest sequence: {}".format(review_len.max()))
print("Shortest sequence: {}".format(review_len.min()))
print("Mean sequence: {}".format(review_len.mean()))
print("Median sequence: {}".format(np.median(review_len)))
print("Total words: {}".format(np.sum(review_len)))
```

```
Longest sequence: 502
Shortest sequence: 1
Mean sequence: 5.397220190197513
Median sequence: 4.0
Total words: 14756
```

In [43]:
```
#create train/test splits
x_train, x_test, y_train, y_test = train_test_split(
        df['review'], df['score'], test_size = 0.2, random_state = 0)
```

In [44]:
```
len(word_cnt(x_train))
```

Out[44]:
```
1953
```

The goal of this tokenization is to break down the OOV words into characters. The primary reason for this is so the raw text can be broken down into tokens which helps the model interpret the meaning of the text by looking at the sequence in an understandable format, (Chakravarthy, 2020).

In [45]:
```
#run tokenizer on training sentences
tokenizer = Tokenizer(num_words=1943, oov_token = '<OOV>')
tokenizer.fit_on_texts(x_train)
```

In [46]:
```
#create vector sequences from sentences
seq_train = tokenizer.texts_to_sequences(x_train)
seq_test = tokenizer.texts_to_sequences(x_test)
```

Padding Process to Standardize the Length of Sequences

To ensure each vector sequence is the same length padding was added to the training and test sequences and truncate sequences longer than the defined maximum length of 13. And while the input text sequences contain a different number of words, it is appropriate. Note: The padding occurs before the text sequence in order to avoid including 0s in the output.

In [47]:
```
#pad and truncate sequences
max_len = 13
seq_train = pad_sequences(seq_train, maxlen=max_len, padding='pre', truncating='post')
seq_test = pad_sequences(seq_test, maxlen=max_len, padding='pre', truncating='post')
```

In [48]:
```
#example sequence
seq_train[0]
```

Out[48]:
```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,  22, 163])
```

Categories of sentiment

Sigmoid activation function will be used for the final dense layer of the network because the sentiment categories are binary, positive (1), negative (0).

In [49]:
```
#categories of sentiment
y_train.value_counts()
```

Out[49]:
```
1    1102
0    1085
Name: score, dtype: int64
```

In [50]:
```
#export dataset and splits
df.to_csv("NLM2_task2_prepared.csv")
```

In summary, to prepare the data for the model, punctuation and special characters were removed, the text was converted to lowercase, tokenization was performed on the “review” column. Next, lemmatization was done to transform words to common roots, and the rare, stop, and neutral words were removed. Finally, the training and test sets were done with an 80/20 split and the sequences of text were tokenized to numeric sequences and the vector sequences were padded and truncated.

In [51]:
```
#build nn model
vocab_len = 1943
dim = 16
max_len=13
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_len, dim, input_length=max_len),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 13, 16)            31088

 global_average_pooling1d (G  (None, 16)               0
 lobalAveragePooling1D)

 dense (Dense)               (None, 6)                 102

 dense_1 (Dense)             (None, 1)                 7

=================================================================
Total params: 31,197
Trainable params: 31,197
Non-trainable params: 0
_________________________________________________________________
```

Embedding - tf.keras.layers.Embedding(vocab_len, dim, input_length=max_len) In the embedding layer, the sequences are represented by 'dim'(16) vectors of 'mex_len'(13) length. Flattening - tf.keras.layers.GlobalAveragePooling1D() The second layer uses

GlobalAveragePoolin1D to flatten the vectors returned by the first layer into a single dimension. Activation(ReLU) - tf.keras.layers.Dense(6, activation='relu') The first activation layer uses the rectified linear unit activation(ReLU) function, which outputs the value or 0, whichever is greater. I've set this layer to use 6 nodes. Activation(Sigmoid) - tf.keras.layers.Dense(1, activation='sigmoid') The sigmoid activation layer ensures that the output is always 1 or 0. The sigmoid activation layer uses 1 node.

In [53]:
```
#validatio
num_epochs=20
history = model.fit(seq_train, y_train, epochs=num_epochs, validation_data = (seq_test
```

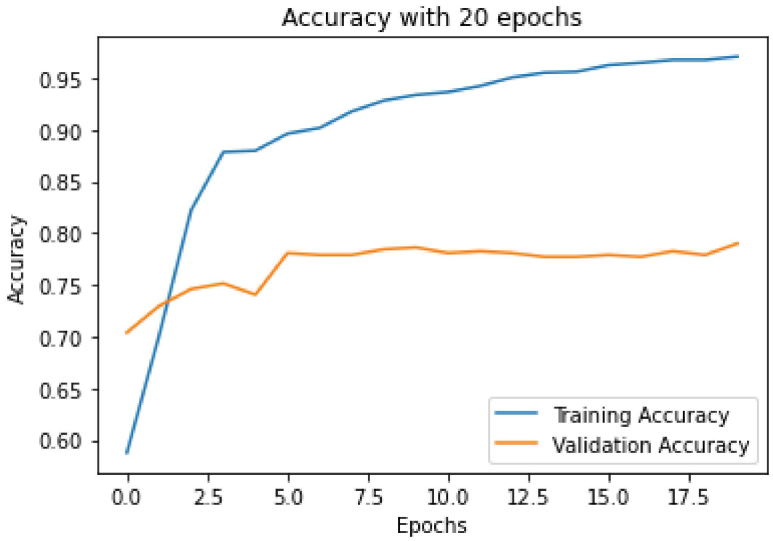
```
Epoch 1/20
69/69 [==============================] - 3s 3ms/step - loss: 0.6917 - accuracy: 0.587
6 - val_loss: 0.6905 - val_accuracy: 0.7038
Epoch 2/20
69/69 [==============================] - 0s 1ms/step - loss: 0.6847 - accuracy: 0.700
5 - val_loss: 0.6827 - val_accuracy: 0.7294
Epoch 3/20
69/69 [==============================] - 0s 1ms/step - loss: 0.6637 - accuracy: 0.822
1 - val_loss: 0.6609 - val_accuracy: 0.7459
Epoch 4/20
69/69 [==============================] - 0s 1ms/step - loss: 0.6175 - accuracy: 0.878
4 - val_loss: 0.6197 - val_accuracy: 0.7514
Epoch 5/20
69/69 [==============================] - 0s 1ms/step - loss: 0.5392 - accuracy: 0.879
7 - val_loss: 0.5715 - val_accuracy: 0.7404
Epoch 6/20
69/69 [==============================] - 0s 1ms/step - loss: 0.4518 - accuracy: 0.896
2 - val_loss: 0.5244 - val_accuracy: 0.7806
Epoch 7/20
69/69 [==============================] - 0s 1ms/step - loss: 0.3765 - accuracy: 0.901
7 - val_loss: 0.4982 - val_accuracy: 0.7788
Epoch 8/20
69/69 [==============================] - 0s 1ms/step - loss: 0.3178 - accuracy: 0.917
7 - val_loss: 0.4855 - val_accuracy: 0.7788
Epoch 9/20
69/69 [==============================] - 0s 1ms/step - loss: 0.2740 - accuracy: 0.928
2 - val_loss: 0.4772 - val_accuracy: 0.7843
Epoch 10/20
69/69 [==============================] - 0s 1ms/step - loss: 0.2402 - accuracy: 0.933
7 - val_loss: 0.4762 - val_accuracy: 0.7861
Epoch 11/20
69/69 [==============================] - 0s 1ms/step - loss: 0.2138 - accuracy: 0.936
4 - val_loss: 0.4797 - val_accuracy: 0.7806
Epoch 12/20
69/69 [==============================] - 0s 1ms/step - loss: 0.1929 - accuracy: 0.942
4 - val_loss: 0.4852 - val_accuracy: 0.7824
Epoch 13/20
69/69 [==============================] - 0s 1ms/step - loss: 0.1749 - accuracy: 0.950
6 - val_loss: 0.4898 - val_accuracy: 0.7806
Epoch 14/20
69/69 [==============================] - 0s 1ms/step - loss: 0.1595 - accuracy: 0.955
2 - val_loss: 0.4988 - val_accuracy: 0.7770
Epoch 15/20
69/69 [==============================] - 0s 1ms/step - loss: 0.1498 - accuracy: 0.956
1 - val_loss: 0.5085 - val_accuracy: 0.7770
Epoch 16/20
69/69 [==============================] - 0s 1ms/step - loss: 0.1378 - accuracy: 0.962
5 - val_loss: 0.5199 - val_accuracy: 0.7788
Epoch 17/20
69/69 [==============================] - 0s 1ms/step - loss: 0.1291 - accuracy: 0.964
8 - val_loss: 0.5283 - val_accuracy: 0.7770
Epoch 18/20
69/69 [==============================] - 0s 1ms/step - loss: 0.1209 - accuracy: 0.967
5 - val_loss: 0.5393 - val_accuracy: 0.7824
Epoch 19/20
69/69 [==============================] - 0s 1ms/step - loss: 0.1134 - accuracy: 0.967
5 - val_loss: 0.5498 - val_accuracy: 0.7788
Epoch 20/20
69/69 [==============================] - 0s 1ms/step - loss: 0.1068 - accuracy: 0.970
7 - val_loss: 0.5634 - val_accuracy: 0.7898
```

In [52]:
```
#sequential model
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_len, dim, input_length=max_len),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

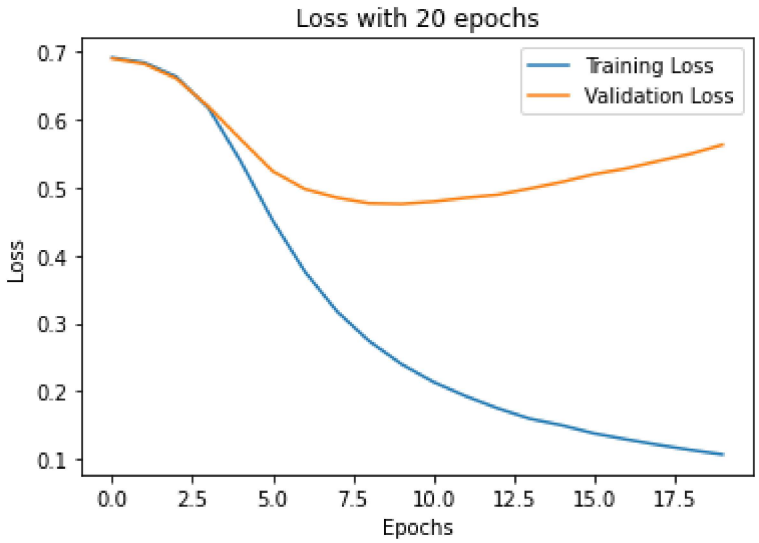
In [54]:
```
#early stop model
stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=1)
num_epochs=20
history_stopped = model.fit(seq_train, y_train, epochs=num_epochs, validation_data = (
```

```
Epoch 1/20
69/69 [==============================] - 0s 2ms/step - loss: 0.1022 - accuracy: 0.969
8 - val_loss: 0.5724 - val_accuracy: 0.7824
Epoch 2/20
69/69 [==============================] - 0s 1ms/step - loss: 0.0982 - accuracy: 0.971
2 - val_loss: 0.5860 - val_accuracy: 0.7879
Epoch 2: early stopping
```

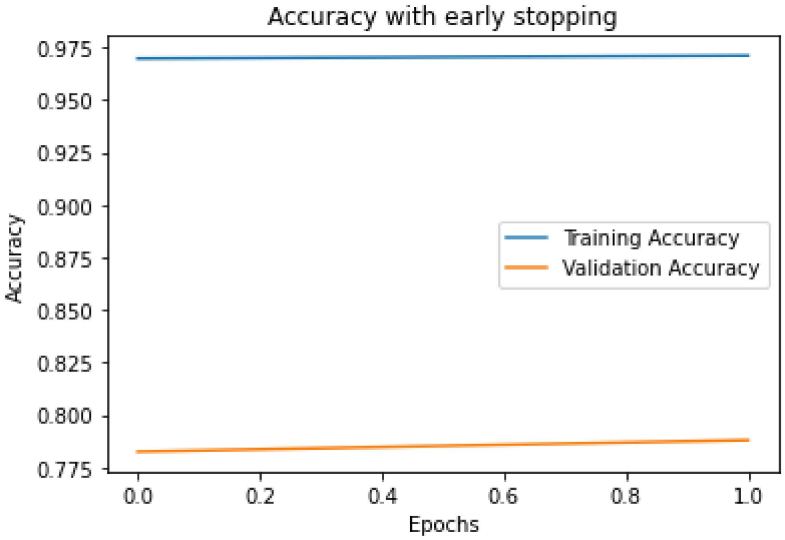
In [55]:
```
#plot accuracy for first run
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Accuracy with 20 epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(["Training Accuracy", "Validation Accuracy"])
plt.show()
```



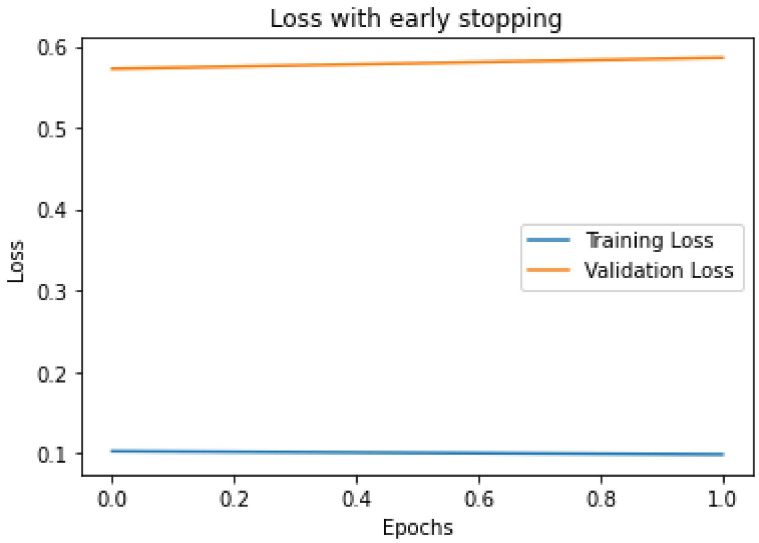
In [56]:
```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("Loss with 20 epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Training Loss", "Validation Loss"])
plt.show()
```



In [57]:
```
# plot accuracy for early stopped run
plt.plot(history_stopped.history['accuracy'])
plt.plot(history_stopped.history['val_accuracy'])
plt.title("Accuracy with early stopping")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(["Training Accuracy", "Validation Accuracy"])
plt.show()
```



In [58]:
```
plt.plot(history_stopped.history["loss"])
plt.plot(history_stopped.history["val_loss"])
plt.title("Loss with early stopping")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Training Loss", "Validation Loss"])
plt.show()
```



In [59]:
```
#predictive accuracy
loss, accuracy = model.evaluate(seq_test, y_test)
```

```
18/18 [==============================] - 0s 589us/step - loss: 0.5860 - accuracy: 0.7
879
```

In [60]:
```
print("Accuracy: {:.2%}".format(accuracy))
```

```
Accuracy: 78.79%
```

Choice of hyperparameters Justification

Activation Functions Sigmoid is appropriate as a final layer for binary classification (Karakaya, 2020) such as this sentiment analysis, as it returns only binary outputs. ReLU can be used to ensure valid inputs to the sigmoid activation layer. Number of Nodes Per Layer For the ReLU layer, I elected to use 6 nodes. This is sufficient for a small dataset to account for input, hidden, and output layers. The final layer only requires 1 node.(Brownlee, 2019) Loss Function The BinaryCrossentropy loss function is appropriate for binary classification.(Karakaya, 2020) Optimizer Adaptive Moment Estimation(ADAM) is the optimizer used, as it is demonstrated to be fast and effective.(Doshi, 2019) Stopping Criteria For stopping criteria, I will aim to minimize validation loss. Using a validation metric as stopping criteria should keep the training from running too long and overfitting the training data. Evaluation Metric The model will be evaluated using the validation loss and validation accuracy to determine how effective it is at classifying review sentiment.

Impact of using stopping criteria vs Defining the Number of Epochs

In order to track value loss, a stopping callback function was used. Initially the training ran for 20 epochs, however, the validation accuracy would drop after 10 epochs. If value started to increase the process was stopped early. The 'patience' was set to 1 in order to avoid the training to stop after the second epoch.

Fitness of the Model and Measures for Overfitting

Because each epoch increases the fit of the model to the training data, stopping early helps to avoid overfitting.

This neural network analysis had a min dim of 16 and max_len of 13 and showed a final percentage of accuracy at 78% with a loss of 46%. Ideally a higher accuracy would have been obtained. At early stopping Epoch 9/20 we saw an accuracy of 93% with a loss of only 24%. Utilizing Adam(Adaptive Moment Estimation) to avoid jumping over the minimum and decrease our velocity kept us at a fairly high accuracy.

My research question for this analysis was "Utilizing sentiment analysis with neural networks for analyzing labeled review sentiments can the sentiments be identified as positive or negative?" Based on the analysis results of 78% I would answer this question as yes with 78% accuracy the neural networks analysis utilizing sentiment analysis can identify sentiments as positive or negative.

References

Chakravarthy, S. (2020, July 10). Tokenization for natural language processing. Medium. Retrieved May 30, 2022, from https://towardsdatascience.com/tokenization-for-natural-language-processing-a179a891bad4

Home. OARC Stats. (n.d.). Retrieved May 19, 2022, from https://stats.oarc.ucla.edu/spss/seminars/efa-spss/#:~:text=Unlike%20factor%20analysis%2C%20principal%20components,into%20common%20a

Doshi, Sanket. "Various Optimization Algorithms For Training Neural Network". towardsdatascience.com, 13 Jan 2019, https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

Karakaya, Murat. "How to solve Binary Classification Problems in Deep Learning with Tensorflow & Keras?". medium.com, 6 Dec 2020 https://medium.com/deep-learning-with-keras/which-activation-loss-functions-part-a-e16f5ad6d82a

Brownlee, Jason. "How to Configure the Number of Layers and Nodes in a Neural Network". machinelearningmaster.com, 6 Aug 2019 https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

In [ ]: