# ****Part I:  Research Question****

## A1: RESEARCH QUESTION

This analysis aims to answer the following research question:

With what level of accuracy does a natural language processing (NLP) neural network trained on the Sentiment Labelled Sentences Data Set (Kotzias, 2015) to predict sentiment from customer reviews, extrapolate to reviews from other data sources?

Answering this question will help inform whether this data set is sufficient for an organization to use for NLP training given its limited size (n=3,000). Likely, the accuracy of the neural network when applied to external data will be poor, indicating the data set is not sufficient for NLP training and has low organizational value.

## A2: OBJECTIVES AND GOALS

**Goal 1**: Develop an NLP neural network capable of accurately classifying positive and negative sentiment from the Sentiment Labelled Sentences Data Set.

* **Objective 1**: Clean the text data of extra characters.
* **Objective 2**: Prepare the data for analysis by removing stop words, lemmatizing, tokenizing, and padding the text.
* **Objective 3**: Build and fit the NLP neural network.

**Goal 2**: Test the accuracy of the neural network from Goal 1 against a sentiment-labeled data set from a different source.

* **Objective 1**: Identify and obtain review data from a second source.
* **Objective 2**: Clean and prepare the data as was done for Goal 1.
* **Objective 3**: Make predictions on the data and assess the accuracy.

## A3: PRESCRIBED NETWORK

To capture the sentiment contained in the text data, a Long Short-Term Memory (LSTM) neural network will be used. LSTM networks work better than standard Recurrent Neural Networks (RNN) with text data because they make use of cell states. This prevents the issues RNNs have with vanishing and exploding gradients due to repeated multiplication of inputs and weights, ultimately extending the memory of the network (Shastri, 2020). With extended memory, entire paragraphs of text can be analyzed in context, allowing sentiment to be captured.

# ****Part II:  Data Preparation****

## B1: DATA EXPLORATION

The text data was assessed for unusual characters. The following characters were found and removed from the text:

!, ", #, $, %, &, ', (, ), \*, +, ,, -, ., /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, :, ;, ?, [, ], ¥, ©, ª, â, ã, –, —, …

A word embedding length of 600 was used. According to the authors of the book, Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps, “…one rule of thumb is to use the fourth root of the total number of unique categorical elements while another is that the embedding dimension should be approximately 1.6 times the square root of the number of unique elements in the category, and no less than 600 (Lakshmanan, Robinson, & Munn, 2020).”

After removing stop words, the final vocabulary size was 4,288 words.

To determine the maximum sequence length to use for the padding process, the word count distribution was reviewed. The maximum length was calculated from the sum of the mean plus three standard deviations, which was approximately 21.87 words. This value was rounded up for a final maximum sequence length of 22.

Chart, histogram

Description automatically generated

## B2: TOKENIZATION

The primary goal of the tokenization process is to split each word into a set of sequences that can be read into a neural network Embedding layer. The inputs are the text review entries, and the outputs are sequences of integers mapped to the words in the vocabulary.

To perform the actual tokenization, the TensorFlow Keras Tokenizer class was fitted and used. Data preparation was performed using pandas, string, and nltk to structure the data, remove special characters, and remove stopwords respectively.

The following code was used to perform tokenization. To allow the code within this report to be executed, imports of the necessary python packages and code to perform data cleaning have also been included. This code was extracted from the file, “task2.ipynb”, which has been included with this submission.

import os

import string

import numpy as np

import pandas as pd

from nltk.corpus import stopwords

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Dense, LSTM

from tensorflow.keras.callbacks import EarlyStopping

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Constants

OUT = 'output2'

PATHS = ['./sentiment\_labelled\_sentences/amazon\_cells\_labelled.txt',

         './sentiment\_labelled\_sentences/imdb\_labelled.txt',

         './sentiment\_labelled\_sentences/yelp\_labelled.txt']

# Load the data

data = []

for path in PATHS:

    with open(path) as file:

        data.extend(file.readlines())

text = [row.split('\t')[0].lower() for row in data]

sentiment = [int(row.split('\t')[1][0]) for row in data]

df = pd.DataFrame().from\_dict({'text': text, 'sentiment': sentiment})

# Remove special characters

characters = string.ascii\_letters + ' '

df.text = df.text.apply(lambda row: ''.join([char for char in row

                                                if char in characters]))

# Remove stopwords

def remove\_stopwords(text: str) -> str:

    stopword\_set = set(stopwords.words('english'))

    temp = text.split()

    for stopword in stopword\_set:

        if stopword in temp:

            temp.remove(stopword)

    return ' '.join(temp)

df.text = df.text.apply(remove\_stopwords)

# Lemmatization

def lemmatize\_text(text: str) -> str:

    lemmatizer = WordNetLemmatizer()

    for pos in ['n', 'v', 'a', 'r', 's']:

        text = ' '.join([lemmatizer.lemmatize(word, pos)

                            for word in text.split()])

    return text

df.text = df.text.apply(lemmatize\_text)

# Tokenization

textt = ' '.join(df.text)

vocab\_size = len(set(textt.split()))

tokenizer = Tokenizer(num\_words=vocab\_size)

tokenizer.fit\_on\_texts(df.text.values)

encoded\_docs = tokenizer.texts\_to\_sequences(df.text.values)

## B3: PADDING PROCESS

Each sequence output by the tokenizer was trimmed or padded to a maximum length of 22 words, which was selected by calculating three standard deviations above the average word count. Each value was rounded up since partial words are not possible in this context. Padding occurred at the end of each sequence where applicable. The following code was used to pad the sequences. It utilizes the TensorFlow Keras pad\_sequences function.

word\_count = [len(x.split()) for x in df.text]

word\_count\_std = np.std(word\_count)

max\_len = int(np.ceil(np.mean(word\_count) + (3 \* word\_count\_std)))

padded\_sequence = pad\_sequences(encoded\_docs, maxlen=max\_len, padding='post')

The following screenshot displays the vocabulary size, the maximum length used for padding, and the first text entry in its cleaned state, in its tokenized state, and finally in its padded state.

Calendar

Description automatically generated

## B4: CATEGORIES OF SENTIMENT

The Sentiment Labelled Sentences Dataset includes a dichotomous representation of sentiment with values of 0 representing negative sentiment and values of 1 representing positive sentiment. Because of this, the neural network will only be able to identify two categories of sentiment. To achieve this binary outcome, the final Dense layer of the neural network will use the sigmoid activation function to produce a probability distribution representing whether each review’s sentiment is more likely positive or negative.

## B5: STEPS TO PREPARE THE DATA

To prepare the data for analysis using an NLP neural network, the following steps were taken. The associated code was presented in sections [B2](#_B2:TOKENIZATION) and [B3](#_B3:PADDING_PROCESS) and has been included with this submission.

1. The data was loaded into a list and parsed into a pandas DataFrame.
2. Special characters were identified and removed from the text using the string package.
3. Stopwords were removed using the Natural Language Toolkit (nltk) library.
4. Each word was lemmatized using the nltk library.
5. The text was tokenized into sequences using the TensorFlow Keras Tokenizer class.
6. The sequences were padded using the TensorFlow Keras padded\_sequence function.
7. The training, validation, and testing datasets were exported to comma-delimited files using the following code.

X\_train, X\_temp, y\_train, y\_temp = \

    train\_test\_split(padded\_sequence, df.sentiment, test\_size=0.4,

                     stratify=df.sentiment, random\_state=42)

X\_val, X\_test, y\_val, y\_test = \

    train\_test\_split(X\_temp, y\_temp, test\_size=0.5,

                     stratify=y\_temp, random\_state=42)

y\_train = np.array(y\_train)

y\_val = np.array(y\_val)

y\_test = np.array(y\_test)

pd.DataFrame(X\_train).to\_csv(os.path.join(OUT, 'X\_train.csv'), index=False)

pd.DataFrame(y\_train).to\_csv(os.path.join(OUT, 'y\_train.csv'), index=False)

pd.DataFrame(X\_val).to\_csv(os.path.join(OUT, 'X\_val.csv'), index=False)

pd.DataFrame(y\_val).to\_csv(os.path.join(OUT, 'y\_val.csv'), index=False)

pd.DataFrame(X\_test).to\_csv(os.path.join(OUT, 'X\_test.csv'), index=False)

pd.DataFrame(y\_test).to\_csv(os.path.join(OUT, 'y\_test.csv'), index=False)

## B6: PREPARED DATASET

The following files have been included with this submission:

1. X\_train.csv
2. y\_train.csv
3. X\_val.csv
4. y\_val.csv
5. X\_test.csv
6. y\_test.csv

# ****Part III:  Network Architecture****

## C1: MODEL SUMMARY

Table

Description automatically generated

## C2: NETWORK ARCHITECTURE

The NLP neural network consists of four layers. There is an Embedding layer that takes the tokenized and padded input, encodes it, and reduces it to the specified embedding length. The second layer is the LSTM layer, which captures the semantic meaning from the text using the cell states of the hidden layers to maintain the history. This is followed by a Dense layer to further process the patterns captured in the LSTM layer. The final layer is a Dense layer with a single node to reduce the output to a binary outcome. The total number of parameters is 2,858,301, all of which are trainable.

## C3: HYPERPARAMETERS

**Activation Functions**

No activation functions were specified for the Embedding or LSTM layers. In the first Dense layer, following the LSTM layer, the Rectified Linear Unit (ReLU) activation function was used. ReLU is a piecewise function made of two linear functions. When x is negative, y equals zero. Otherwise, y equals x. The ReLU activation function performs better in multi-layer networks because it can overcome the vanishing gradients problem (Brownlee, 2020). The final Dense layer uses the sigmoid function because it returns values between 0 and 1 representing the probability distribution for the 0 and 1 sentiment values.

**Number of Nodes Per Layer**

The Embedding layer accepts 22 inputs, used to accept the padded sequences, and outputs 600 nodes containing the embedded values mapped to the 4,288-word vocabulary. The LSTM layer contains 100 nodes. Through trial and error, this proved to be large enough to capture the sentiment in the text without overfitting to the data when other hyperparameters were tuned. The first Dense layer contains 50 nodes. This layer was added to increase the validation accuracy of the model, but still needs to continue to reduce the output, so half the node count of the LSTM model was used. The final Dense layer contains a single node to produce a single outcome for each input sequence.

**Loss Function**

The function used to measure loss during training was Binary Cross-Entropy. Binary Cross-Entropy calculates the natural logarithm of the difference between the predicted value and its true label. This value is then summed across all predictions to obtain a total for the network (Martinek, 2020). Because the natural logarithm is used, cross-entropy acts similarly to mean squared error in that it assigns a higher penalty to larger differences. This ensures the predictions stay as close as possible to their true labels.

**Optimizer**

The neural network was compiled using the Adam optimizer. This optimizer combines the gradient descent concepts of momentum and root mean square propagation (RMSP) to both converge faster and overcome local minima better than most other optimizers today (prakharr0y, 2020). In addition, the Adam optimizer uses fewer computing resources. When training more complicated models, this can become an important factor.

**Stopping Criteria**

To prevent the neural network from overfitting, the early stopping monitor from TensorFlow Keras was added to the compiler’s callbacks. The early stopping monitor watches the network’s loss as training happens and stops it short of the specified number of epochs if enough epochs pass without the network’s loss improving. The number of epochs that are allowed to pass without improvement is controlled by the patience parameter. For this network, patience was set to 2.

**Evaluation Metric**

To evaluate the neural network, the Accuracy metric was used. Accuracy is calculated as the total number of correct predictions divided by the total number of predictions. The goal was to increase this metric as much as possible without overfitting to the training data.

# ****Part IV:  Model Evaluation****

## D1: STOPPING CRITERIA

Stopping criteria is an important part of training a neural network. When a model is fitted to the training data, the initial weights are always initialized randomly, meaning you never know how accurate the model will be during the first epoch. Because of this, it is impossible to know exactly how many epochs are necessary to properly train a neural network. Instead of relying solely on a static number of epochs to run, stopping criteria uses the metrics from each epoch to determine whether to continue fitting and stops early if no improvement is made. This is crucial in preventing overfitting.



## D2:TRAINING PROCESS

Chart, line chart

Description automatically generated

## D3: FIT

The final training and validation loss metrics were 0.1101 and 0.6984, respectively. While the validation loss is notably higher than the training loss, it is improved from the initial attempts to fit the network. This can be seen in the report, task2\_report.html, which has been included with this submission, where the validation loss of the initial network was 1.1214. To reduce overfitting, an early stopping monitor with patience of 2 was added to the network compiler’s callbacks.

## D4: PREDICTIVE ACCURACY

Predictive accuracy was measured using the accuracy score, which is a ratio of the number of correct predictions to the total number of predictions. The final accuracy score, calculated using the testing data, was 0.78. This can be seen in the screenshot and confusion matrix below.

Shape, rectangle

Description automatically generated

Chart, treemap chart

Description automatically generated

Additionally, to answer the research question, a secondary sentiment-labeled data set was obtained. The Google Local Data (2021) data offers dense subsets of local review data by US state (Li, Shang, & McAuley, 2022) (Yan, He, Li, Zhang, & Mcauley, 2022). From the Michigan dense subset, 20,000 records were downloaded. After dropping records with null values, 12,534 records remained. The review scores were done on a scale of 1 to 5 with 1 being the lowest and 5 being the highest. Scores of 1 or 2 were mapped to negative sentiment or 0. Scores of 4 or 5 were mapped to positive sentiment or 1. Scores of 3 were considered neutral and dropped. This produced a final data set with 10,339 reviews labeled positive and 1,713 reviews labeled negative. After making predictions on the Google review data set, the accuracy score was 0.8255. This can be seen in the screenshots and confusion matrix below.





Chart, treemap chart

Description automatically generated

This score indicates that, while the Sentiment Labelled Data Set may be relatively small, it is surprisingly representative of review data in general.

# ****Part V:  Summary and Recommendations****

embedding\_vector\_length = 600

early\_stopping\_monitor = EarlyStopping(patience=2)

model2 = Sequential()

model2.add(Embedding(vocab\_size,

                     embedding\_vector\_length,

                     input\_length=max\_len))

model2.add(LSTM(100))

model2.add(Dense(50, activation='relu'))

model2.add(Dense(1, activation='sigmoid'))

model2.compile(optimizer='adam',

               loss='binary\_crossentropy',

               metrics=['accuracy'])

history2 = model2.fit(X\_train, y\_train, epochs=20, batch\_size=32,

                      validation\_data=[X\_val, y\_val],

                      callbacks=[early\_stopping\_monitor])

model2.save(os.path.join(OUT, 'fitted\_model.h5'))

## E: CODE

## F: FUNCTIONALITY

The final neural network is sequential. It starts by embedding the encoded sequences mapped to the 4,233-word vocabulary into 600 outputs. The outputs, multiplied by their weights, are fed into the LSTM layer. This computes the cell states one step at a time, performing several non-linear computations in the process. The outputs of the LSTM layer, multiplied by their weights, are fed into the Dense, 50-node layer. There, the ReLU activation function transforms the negative values to zeroes. The outputs of the first Dense layer, multiplied by their weights, are fed into the final layer which is Dense and contains a single node. This node performs the final activation function, sigmoid, which converts the values to the probability distribution representing the likelihood that the review was positive or negative.

By including the embedding layer, the information is first condensed, which improves the time and processing performance of the network. The LSTM layer is crucial for identifying sentiment and is the primary NLP component of the network. The Dense layer was added between the LSTM and output layers to help increase validation accuracy scores by identifying deeper patterns from the cell states than if they had been fed straight into the final output layer.

## G: RECOMMENDATIONS

Based on the accuracy scores observed in the testing and secondary data, this model does contain some value for training an NLP neural network. While I would not recommend placing a model achieving an 82% accuracy score into a high-stakes production environment, this data could be appended to additional data to make a very robust network.

# ****Part VI: Reporting****

## H: REPORTING

The HTML document of the executed notebook has been included with this submission as task2\_report.html.

## I: SOURCES FOR THIRD-PARTY CODE

Kotzias, D. (2015, May 30). *Sentiment Labelled Sentences Data Set.* Retrieved from UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences#

Li, J., Shang, J., & McAuley, J. (2022). UCTopic: Unsupervised Contrastive Learning for Phrase Representations and Topic Mining. *Annual Meeting of the Association for Computational Linguistics (ACL).* Retrieved from jiachengli1995.github.io.

Yan, A., He, Z., Li, J., Zhang, T., & Mcauley, J. (2022). Personalized Showcases: Generating Multi-Modal Explanations for Recommendations. *arXiv:2207.00422*.

## J: SOURCES

Brownlee, J. (2020, August 20). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Retrieved from MachineLearningMastery.com: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Lakshmanan, V., Robinson, S., & Munn, M. (2020). *Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps.* Sebastopol: O'Reilly Media.

Martinek, V. (2020, May 22). *Cross-entropy for classification. Binary, multi-class and multi-label classification*. Retrieved from Towards Data Science: https://towardsdatascience.com/cross-entropy-for-classification-d98e7f974451

prakharr0y. (2020, October 24). *Intuition of Adam Optimizer*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/

Shastri, A. (2020, November 24). *3 neural network architectures you need to know for NLP!* Retrieved from Towards Data Science: https://towardsdatascience.com/3-neural-network-architectures-you-need-to-know-for-nlp-5660f11281be