

# NLM3 — Task 2: Sentiment Analysis Using Neural Networks

## Part I: Research Question

### A. Describe the purpose of this data analysis by doing the following:

1. Can we automatically determine the sentiment (either positive, or negative) of a future reviews left online, so that they can automatically be directed to the appropriate departments for follow up when necessary.
2. The goal is to build a neural network, trained on previous reviews the company has recieved across 2 different websites (amazon and yelp). This Neural Network will have the ability to determine the sentiment of a future review left on on of these sites.
3. A Recurrent Neural Network or RNN can be used to determine the binary sentiment of a text string after being trained on labeled data. Since we already have labeled data, this is a good option for what we are needing to do.

## Part II: Data Preparation

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import re
import math
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 Dense, Embedding, LSTM
from tensorflow.keras.callbacks import EarlyStopping
```

```
In [2]: amazon_df = pd.read_csv('amazon_cells_labelled.txt', delimiter='\t', header=
yelp_df = pd.read_csv('yelp_labelled.txt', delimiter='\t', header=None)

df = pd.concat([amazon_df, yelp_df], ignore_index=True)
df.columns = ['review', 'score']
```

```
with pd.option_context('display.max_rows', 5, 'display.max_columns', None):
    display(df)
```

	review	score
0	So there is no way for me to plug it in here i...	0
1	Good case, Excellent value.	1
...	...	...
1998	The whole experience was underwhelming, and I ...	0
1999	Then, as if I hadn't wasted enough of my life ...	0

2000 rows × 2 columns

## B. Summarize the data cleaning process by doing the following:

1. Perform exploratory data analysis on the chosen data set, and include an explanation of each of the following elements:

- presence of unusual characters (e.g., emojis, non-English characters)

Letters, numbers, and punctuation are all expected characters in reviews posted online. The neural network should be able to handle these typical english characters well. But often punctuation marks do not contribute significant value to the sentiment of the phrase, so to simplify the analysis and better facilitate proper results, we are going to remove them. Additionally, any non english characters or non literary characters like emojis would clash with the otherwise english alphabet the neural network will be using. So we should identify these other characters and remove them too. Ultimately this means only letters and numbers will remain in our text set.

```
In [3]: other_characters = set()
for review in df['review']:
    other_characters.update(re.findall(r'[^a-zA-Z0-9\s]', review))

other_characters.add('\t\n')
other_characters = ''.join(sorted(list(other_characters)))
print(other_characters)

# df['review'] = df['review'].apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]+', '',
```

!"#\$%&'()\*+,-./:;?[ ]éê

- vocabulary size

Using the tf tokenizer we can split up our text set into a vocabulary. We are removing all of the non english characters that our previous step reported having found, we are lowercasing everything for standardization, we are splitting on the space character, and we are replacing the unknown characters (mostly numbers) with the symbol.

```
In [4]: tokenizer = Tokenizer(
        filters=other_characters,
        lower=True,
        split=' ',
        oov_token='<OOV>',
    )
    tokenizer.fit_on_texts(df['review'])
    word_count = tokenizer.word_index

    vocab_size = len(word_count) + 1

    print('Size of Vocabulary:', vocab_size)
    print('Sampling of Words:', list(word_count.items())[15:20])
```

Size of Vocabulary: 3230

Sampling of Words: [('with', 16), ('very', 17), ('t', 18), ('good', 19), ('great', 20)]

- proposed word embedding length

There are a number of different ways to determine the embedding length, and really it should be optimized for to find the best embedding length for your given vector space and vocabulary. Some sources suggest simply using 100 dimensions. Others say to take the 4th square of your vocabulary size. We can try with both. The fourth square of my vocabulary size (rounded up) is 8.

```
In [5]: embedding_length = int(np.ceil(np.sqrt(np.ceil(np.sqrt(vocab_size)))))

    print('Embedding Length:', embedding_length)
```

Embedding Length: 8

- statistical justification for the chosen maximum sequence length

In order to maintain as much information as possible from the input dataset, I am going to elect to keep as much information as possible, and use padding to normalize my vector space for the shorter sentences. Which means my maximum sequence length will be 32.

```
In [6]: max_sentence_length = 0
    min_sentence_length = math.inf
    for review in df['review']:
        if len(review.split()) > max_sentence_length:
            max_sentence_length = len(review.split())
        if len(review.split()) < min_sentence_length:
```

```
min_sentence_length = len(review.split())

print('Max Sentence Length:', max_sentence_length)
print('Min Sentence Length:', min_sentence_length)
```

Max Sentence Length: 32

Min Sentence Length: 1

2. Describe the goals of the tokenization process, including any code generated and packages that are used to normalize text during the tokenization process.

The goal of the tokenization process is to reduce the words from the text input set into numerical representations. I used the `tf.keras` tokenizer package to assign indexes to each unique word inside my dataset. The collection of these unique words makes up the vocabulary of my dataset. All of the code to do this can be seen above.

3. Explain the padding process used to standardize the length of sequences. Include the following in your explanation:

To produce the padding matrix I used the `pad_sequences` function from the `tf.keras` library again. This utilized the already produced word indexes and the predetermined max length of the sequences to build a matrix of numbers. Where each row in the matrix represents 1 review. And each entry in that row represents 1 word from that review. For reviews shorter than the decided 32 length, 0s are used to fill in (or pad) the end of the row. An example of one of these number transformed and padded sequences can be seen below.

```
In [7]: padding = pad_sequences(
    tokenizer.texts_to_sequences(df['review']),
    maxlen=max_sentence_length,
    padding='post',
    truncating='post',
)

print('Padding Shape:', padding.shape)
print('Padding Sample:', padding[0])
```

Padding Shape: (2000, 32)

Padding Sample: [ 29 56 8 59 143 13 64 7 271 5 15 48  
15 2  
150 451 3 63 114 6 1418 0 0 0 0 0 0  
0 0 0 0]

4. Identify how many categories of sentiment will be used and an activation function for the final dense layer of the network.

Displayed below, the total number of sentiment outputs labeled in our dataset is 2. 0 meaning negative and 1 meaning positive. The activation function we will use is the

Sigmoid function. It is an industry standard for binary outputs and works well with NLP problems as it returns a probability of labeling which is most reasonable for the likely outputs of an NLP problem.

```
In [8]: outputs = df['score'].astype('int32').unique()
print('Outputs:', f'({len(outputs)})', outputs)
num_outputs = len(outputs)
```

Outputs: (2) [0 1]

5. Explain the steps used to prepare the data for analysis, including the size of the training, validation, and test set split (based on the industry average).

The Steps used to prepare the data for training are as follows:

- Import the datasets from their respective files and read them in as pandas dataframes.
- Validate the data by ensuring that the outputs are within the expected scope, and that the inputs are made up of only the expected characters.
- Remove any unwanted characters from the set of inputs.
- Cast the inputs all to lowercase so that cases don't mess up the tokenization process.
- Tokenize the inputs into numerical "word index" representations.
- Apply padding to the tokenized inputs so that they can all be the same shape.
- Split the Dataset into a Training set and a Testing set.

This last step is seen below, where we split the inputs and their corresponding outputs into 2 groups. A training set that is 80% of the data. And a test set that is the other 20%.

```
In [9]: training_size = int(len(df) * 0.8)
training_input_padded = padding[:training_size]
training_output = df['score'][:training_size].values.reshape(-1, 1)
testing_input_padded = padding[training_size:]
testing_output = df['score'][training_size:].values.reshape(-1, 1)
```

6. Provide a copy of the cleaned data set.

Cleaned dataframe ready for analysis will be included in the upload by the name of `cleaned_data.csv`

```
In [10]: df['review'] = padding.tolist()
df.to_csv('cleaned_data.csv', index=False) # Save to a CSV file
```

## Part III: Network Architecture

```
In [11]: # Model
         stopping = EarlyStopping(
             monitor='val_accuracy',
             patience=3,
             verbose=0,
             mode='auto',
             restore_best_weights=True
         )
         model = Sequential([
             Embedding(vocab_size, 100),
             LSTM(64, dropout=0.5),
             Dense(1, activation='sigmoid'),
         ])
         model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

         history = model.fit(
             training_input_padded,
             training_output,
             epochs=15,
             validation_split=0.2,
             callbacks=[stopping],
             verbose=1
         )
```

```

Epoch 1/15
40/40 ██████████ 1s 12ms/step - accuracy: 0.4580 - loss: 0.6965 - val_accuracy: 0.5594 - val_loss: 0.6875
Epoch 2/15
40/40 ██████████ 0s 9ms/step - accuracy: 0.5028 - loss: 0.6944 - val_accuracy: 0.6000 - val_loss: 0.6900
Epoch 3/15
40/40 ██████████ 0s 9ms/step - accuracy: 0.6260 - loss: 0.6408 - val_accuracy: 0.7531 - val_loss: 0.5281
Epoch 4/15
40/40 ██████████ 0s 9ms/step - accuracy: 0.8703 - loss: 0.3326 - val_accuracy: 0.7625 - val_loss: 0.6167
Epoch 5/15
40/40 ██████████ 0s 9ms/step - accuracy: 0.9478 - loss: 0.1698 - val_accuracy: 0.7656 - val_loss: 0.5646
Epoch 6/15
40/40 ██████████ 0s 9ms/step - accuracy: 0.9588 - loss: 0.1382 - val_accuracy: 0.7688 - val_loss: 0.7523
Epoch 7/15
40/40 ██████████ 0s 9ms/step - accuracy: 0.9794 - loss: 0.0864 - val_accuracy: 0.7719 - val_loss: 0.6480
Epoch 8/15
40/40 ██████████ 0s 9ms/step - accuracy: 0.9820 - loss: 0.0583 - val_accuracy: 0.7531 - val_loss: 0.7956
Epoch 9/15
40/40 ██████████ 0s 9ms/step - accuracy: 0.9835 - loss: 0.0619 - val_accuracy: 0.7812 - val_loss: 0.7464
Epoch 10/15
40/40 ██████████ 0s 9ms/step - accuracy: 0.9908 - loss: 0.0414 - val_accuracy: 0.7594 - val_loss: 0.7786
Epoch 11/15
40/40 ██████████ 0s 9ms/step - accuracy: 0.9899 - loss: 0.0442 - val_accuracy: 0.7688 - val_loss: 0.9713
Epoch 12/15
40/40 ██████████ 0s 9ms/step - accuracy: 0.9807 - loss: 0.0644 - val_accuracy: 0.7563 - val_loss: 1.1928

```

### C. Describe the type of network used by doing the following:

1. Provide the output of the model summary of the function from TensorFlow.

```
In [12]: model.summary()
```

**Model: "sequential"**

Layer (type)	Output Shape	Par
embedding ( <a href="#">Embedding</a> )	(32, 32, 100)	323
lstm ( <a href="#">LSTM</a> )	(32, 64)	42
dense ( <a href="#">Dense</a> )	(32, 1)	

**Total params:** 1,095,917 (4.18 MB)  
**Trainable params:** 365,305 (1.39 MB)  
**Non-trainable params:** 0 (0.00 B)  
**Optimizer params:** 730,612 (2.79 MB)

## 2. Discuss the number of layers, the type of layers, and the total number of parameters.

The RNN network that I used for this analysis consists of 3 layers.

1. The embedding layer that creates the embedding vectors out of the tokenized text inputs that are passed to it. Its input shape is made up of the 32 reviews sent to it simultaneously because of the batching (default value) and the max review length of 32 that all other reviews reached with padding. Its output shape is the exact same, except for now each token has an additional layer of depth to it, being expanded out by 100 dimensions from the embedding process. This is what is referred to by the "Embedding Length" (Kwan, 2023)
2. The LSTM layer is the real meat of the RNN. This is the layer with all of the hidden nodes. Its doing all of the analysis on the embedding vectors to make predictions. LSTM stands for Long Short Term Memory. Which refers to a short term sequence memory being applied over long gaps in the sequence. Thus long-short-term memory. In this layer I have put 64 different node. Which is the cause for the result of 64 output numbers per review being passed in. These 64 nodes that make up the hidden layers are individually used only half of the time, since I am using a 50% dropout parameter. This makes it so that no single node is overly powerful in its weights applied, and ensure that the model does not over fit to the training data. (Virahonda, 2020)
3. Finally the last layer is a dense layer, which is used to reduce the response down to a single value, the last layer uses a sigmoid activation function to condense all of the input values down into 1, the output.

The total number of parameters is 1,095,917 which means that across all of the embeddings that are passed through the multiple layers both explicit and hidden, 1,095,917 weights and biases got applied at the various nodes.

I did decide to only use 1 hidden layer here, with the LSTM layer. Its very common for RNN networks to only have a few hidden layers, and in testing with upwards of 3 hidden layers, I didn't see any additional gains over just the single layer. So for the sake of keeping the network light while also ensuring accuracy, we are just going to use 1 hidden LSTM layer.

## 3. Justify the choice of hyperparameters, including the following elements:



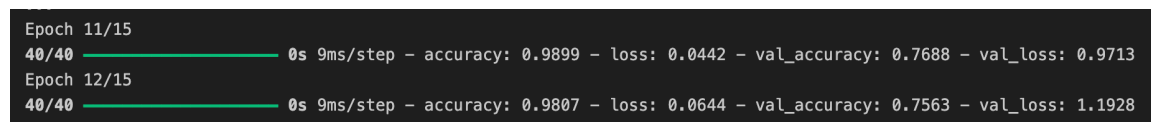
- activation functions
  - The only activation function that I left in my network was the Sigmoid activation in the output layer. This is considered across the industry as the best function for condensing to a binary value. I did try out using a relu activation function between my hidden layers, but that seemed to poorly impact accuracy so it was removed.
- number of nodes per layer
  - This is an interesting metric to play with, cause there is no "perfect" number to place here, It has to be discovered empirically. So thats how I did it. I set the model inside of a loop that ran performance checks at intervals of 2 digit jumps for the count of nodes in the hidden layer. It demonstrated decreasing accuracy as the size of the hidden layer nodes grew. Too small too was no good as I decided to dropout half of the nodes randomly each run in order to mitigate the chances of overfitting. The sweetspot was I ended up with was 64 Nodes.
- loss function
  - With a binary output really the best and only choice in loss functions is the `binary_crossentropy` function. It is the best for that use case.
- optimizer
  - Adam is the best optimizer. And that doesn't even really need a defense, Its widely accepted as the best and used as such. It uses the best gradient decent for optimizing the direction of the network.
- stopping criteria
  - I did use a stopping criteria callback function to prevent wasted compute on epochs with likely worse performance. I used a patience window of 3 which, if anything, that was too big. 2 would have sufficed, but I wanted to be extra sure, and a singular extra epoch is not impactful enough to the compute cost or speed to really make any difference.
- evaluation metric
  - For the evaluation metric we kept things simple. Just straight up Accuracy. A measure of the percentage of the inputs, that with the current weights in the network would predict the same value as their labels. For training purposes across the epochs however, this accuracy number had to be based on the validation set, and not the training set. This helps to further minimize the likely hood of overfitting.

## Part IV: Model Evaluation

### D. Evaluate the model training process and its relevant outcomes by doing the following:

1. Discuss the impact of using stopping criteria to include defining the number of epochs, including a screenshot showing the final training epoch.

The biggest expense in training an NLP model is the expense of Compute. It takes a lot of power to iterate in a million different ways across all of the different models and fine tune their hyper parameters. One of the easiest ways to save on that compute cost and speed up the training process is to cut it short early. Since the models train in sequential epochs, the best way to cut it short is to just not allow it to continue to the next epoch if it appears to not be improving any further. There really is no perfect way of knowing when it is not improving further, but the way that I used is to stop the epochs after 3 epochs in a row fail to improve on the best validation accuracy metric the previous epoch had achieved. This does leave us with the possibility of falling into a local minima, but the chances of that are small enough and mostly handled by the use of Validation Accuracy as the stopping metric that the saved time and costs outweigh the possible loss from a local minima stop. I almost never saw any improvements in accuracy outside of ~15 epochs, so as a final stopping criteria I set 15 as the max number of epochs my model would execute. In the training fit, my model got to 12 epochs before the accuracy criteria stopped it short. The screen shot of the last 2 epochs' results are attached.



The screenshot shows the training progress for two epochs. For Epoch 11/15, the progress bar is at 40/40, and the metrics are: 0s 9ms/step - accuracy: 0.9899 - loss: 0.0442 - val\_accuracy: 0.7688 - val\_loss: 0.9713. For Epoch 12/15, the progress bar is also at 40/40, and the metrics are: 0s 9ms/step - accuracy: 0.9807 - loss: 0.0644 - val\_accuracy: 0.7563 - val\_loss: 1.1928.

Epoch	Progress	Time	Step Time	Accuracy	Loss	Val Accuracy	Val Loss
11/15	40/40	0s	9ms/step	0.9899	0.0442	0.7688	0.9713
12/15	40/40	0s	9ms/step	0.9807	0.0644	0.7563	1.1928

2. Assess the fitness of the model and any actions taken to address overfitting.

A number of steps were taken to avoid overfitting that have already been mentioned. Namely, that accuracy was measured through the validation set the entire training process and that dropouts were used in the hidden layers. The confirmation of this is seen in the testing of model with the left aside testing set of data (separated from both the training data and the validation data) As seen below, the testing results are as good if not better than the validation results. Which means we did a really good job at avoiding overfitting, and ultimately achieved 80% accuracy with data not at all involved in the model creation. This high accuracy with the testing data indicates a proper pattern recognition, meaning that it also did not under fit from the training data, but was able to predict sentiments correctly over 80% of the time. Much better than the 50% of the time that would have happened with just random chance.

```
In [17]: results = model.evaluate(testing_input_padded, testing_output)

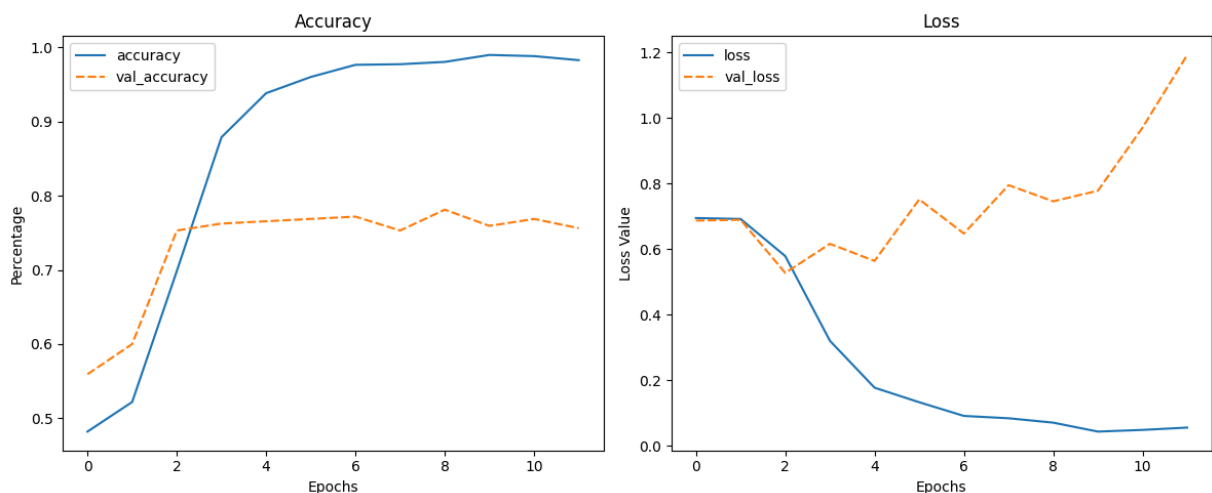
print('Test Loss:', results[0])
print('Test Accuracy:', results[1])
```

**13/13** ————— **0s** 5ms/step – accuracy: 0.7945 – loss: 0.7902  
 Test Loss: 0.669361412525177  
 Test Accuracy: 0.8075000047683716

3. Provide visualizations of the model's training process, including a line graph of the loss and chosen evaluation metric.

```
In [18]: history_df = pd.DataFrame(history.history)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))
sns.lineplot(data=history_df[['accuracy', 'val_accuracy']], ax=axes[0])
sns.lineplot(data=history_df[['loss', 'val_loss']], ax=axes[1])
axes[0].set_title('Accuracy')
axes[1].set_title('Loss')
axes[0].set_xlabel('Epochs')
axes[1].set_xlabel('Epochs')
axes[0].set_ylabel('Percentage')
axes[1].set_ylabel('Loss Value')
plt.tight_layout()
plt.show()
```



4. Discuss the predictive accuracy of the trained network using the chosen evaluation metric from part D3.

The validation accuracy was the metric I chose to optimize my model towards. It is produced using the quotient of the number of reviews that the model would have predicted correctly with the total number of validation reviews tested. The validation data is involved in the training process since the accuracy of it is computed each epoch and the continuation of the epochs is determined by its most recent calculation of the validation accuracy with the model weights as they are currently set. But It is not used in

the actual training and determining of the weights at each node. So since it is removed from the actual training, but not so much so as to be uninvolved in the training, it is a very good metric to use for Predictive accuracy and getting an honest picture of how that looks.

Ultimately the accuracy that we care about though is the predictive accuracy of unknown and unlabeled data. To synthesize this, I set aside a portion of the data as a Testing Set. Running this data through the already trained model gave me an accuracy measure of how the model performs predictably on unseen data, and it did great. >80% accuracy!

## Part V: Data Summary and Implications

### E. Provide the code you used to save the trained network within the neural network.

```
In [19]: model.save('best_model.keras')
```

### F. Discuss the functionality of your neural network, including the impact of the network architecture.

The Neural Network is initially powered by the input data coming from over 2000 reviews from the Amazon and Yelp platforms about our company and the products that we sell. Those reviews were labeled with their sentiment (either positive or negative) and the model was optimized to produce outcomes that coincided with the provided labels. This makes the network capable of producing a sentiment label for reviews that were even not included in the training. The network architecture used to achieve this was the Keras API built into the TensorFlow package. That API allowed for easy use of common NLP functions that allowed me to build the RNN network that I did. It also allowed for the network to be optimized in training across the hyperparameter inputs and model design (number and types of layers). All of this architecture was ultimately pointed at the goal of accurate predictive capacity. And it appears to have achieved that well.

### G. Recommend a course of action based on your results.

Referring back to the original questions and goal of this analysis, I can now confidently answer that yes, it is possible to produce a model that can automatically assign a sentiment label to future reviews that are received. The model that we were able to produce works very well in labeling reviews accurately.

Now that we know that, the course of action I would recommend is to set up a pipeline for new reviews that are received, so that their text content automatically gets evaluated with this model. Based on the assigned label, the review can then get forwarded to either

the marketing department for a case study support (positive sentiment) or the the customer support team for triaging (negative sentiment).

## Part VI: Reporting

### H. This Notebook

[https://colab.research.google.com/drive/11awhUENecmwxPD4t\\_fAp2ySJ5DWX2DU?usp=sharing](https://colab.research.google.com/drive/11awhUENecmwxPD4t_fAp2ySJ5DWX2DU?usp=sharing)

### I/J. Works Cited

9.4. recurrent neural networks. 9.4. Recurrent Neural Networks - Dive into Deep Learning 1.0.3 documentation. (n.d.). [https://d2l.ai/chapter\\_recurrent-neural-networks/rnn.html](https://d2l.ai/chapter_recurrent-neural-networks/rnn.html)

Kwan, M. (2023, September 6). Finding the optimal number of dimensions for word embeddings. Medium. <https://medium.com/@matti.kwan/finding-the-optimal-number-of-dimensions-for-word-embeddings-f19f71666723>

szli, & Andy Hayden. (1958, February 1). How to reshape a pandas.series. Stack Overflow. <https://stackoverflow.com/questions/14390224/how-to-reshape-a-pandas-series>

Tamim Addari, & Daniel De Freitas. (1961, December 1). How to stack multiple LSTM in Keras?. Stack Overflow. <https://stackoverflow.com/questions/40331510/how-to-stack-multiple-lstm-in-keras>

Team, K. (n.d.-a). Keras documentation: Embedding layer. [https://keras.io/api/layers/core\\_layers/embedding/](https://keras.io/api/layers/core_layers/embedding/)

Team, K. (n.d.-b). Keras Documentation: LSTM Layer. [https://keras.io/api/layers/recurrent\\_layers/lstm/](https://keras.io/api/layers/recurrent_layers/lstm/)

Team, K. (n.d.-c). Keras Documentation: Model training apis. [https://keras.io/api/models/model\\_training\\_apis/#evaluate-method](https://keras.io/api/models/model_training_apis/#evaluate-method)

Team, K. (n.d.-d). Keras Documentation: Modelcheckpoint. [https://keras.io/api/callbacks/model\\_checkpoint/](https://keras.io/api/callbacks/model_checkpoint/)

Tf.keras.utils.pad\_sequences : tensorflow V2.16.1. TensorFlow. (n.d.). [https://www.tensorflow.org/api\\_docs/python/tf/keras/utils/pad\\_sequences](https://www.tensorflow.org/api_docs/python/tf/keras/utils/pad_sequences)

Virahonda, S. (2020, October 10). An easy tutorial about sentiment analysis with deep learning and keras. Medium. <https://towardsdatascience.com/an-easy-tutorial-about-sentiment-analysis-with-deep-learning-and-keras-2bf52b9cba91>

Wikimedia Foundation. (2024, December 12). Long short-term memory. Wikipedia.  
[https://en.wikipedia.org/wiki/Long\\_short-term\\_memory](https://en.wikipedia.org/wiki/Long_short-term_memory)