Gurjus Singh

November 8^{th}, 2020

MSDS 422 Practical Machine Learning

Assignment #8 Language Modeling with an RNN

Data preparation, exploration, visualization

The goal of this Data Analysis was to use two types of Neural Networks specifically Recurrent Neural Networks and Long short-term memory Neural Networks for **Binary Classification** on IMBD Movie Reviews. This is considered a supervised learning method as there was a target variable in the Movies dataset. Before implementing the models, I first had to download the dataset and split it using tf.load function from TensorFlow package. The dataset had its own encoder that was used to encode the text to numbers. I got an idea of how it encodes by showing encoding a sample string shown in output 1-1. I then saw how the encoded string was mapped to each text pattern in output 1-2. Next I wanted to seem of the text and get an idea of the types of text the encoder encodes from the movie reviews dataset. I saw this my looking at the plain text dataset seen in Output 1-3. From the text negative reviews were seen through the words 'boring', 'worst', 'bad' while the positive reviews were dictated by the words of 'good', 'best', 'great' , and 'entertaining'.

```
Encoded string is [4025, 222, 4277, 4413, 878, 1848, 2675, 2975, 2509, 6623, 8044, 7975]
The original string: "Hello Northwestern Data Science Students."
```

Output 1-1

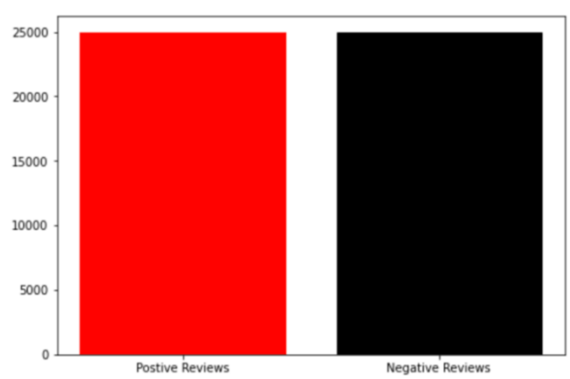
```
4025 ----> Hell
222 ----> o
4277 ----> North
4413 ----> western
878 ----> Da
1848 ----> ta
2675 ----> Sci
2975 ----> ence
2509 ----> Stu
6623 ----> dent
8044 ----> s
7975 ----> .
```

Output 1-2

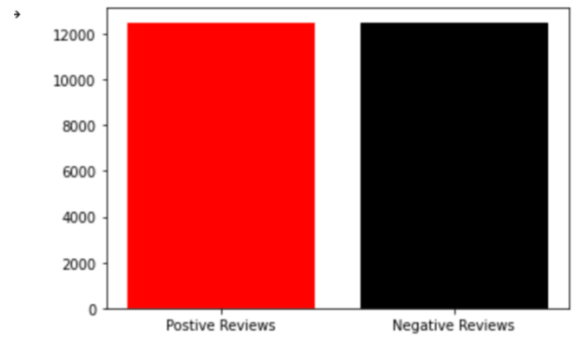
```
array([b"There are films that make careers. For George Romero, it was NIGHT OF THE LIVING DEAD; for Kevin Smith, CLERKS; for Robert Rodriguez, EL MARIACHI. Add to that list Onur Tukel's
       b"A blackly comic tale of a down-trodden priest, Nazarin showcases the economy that Luis Bunuel was able to achieve in being able to tell a deeply humanist fable with a minimum of
       b'Scary Movie 1-4, Epic Movie, Date Movie, Meet the Spartans, Not another Teen Movie and Another Gay Movie. Making "Superhero Movie" the eleventh in a series that single handily
       ...,
       b"Most predicable movie I've ever seen...extremely boring, I feel like I've seen a hundred movies with the same storyline as this one. Acting is OK at best, there's no action real
       b"It's exactly what I expected from it. Relaxing, humorous and entertaining. The acting couple was awesome, as well as the scene selection. I personally recommend this. It's kind
       b"They just don't make cartoons like they used to. This one had wit, great characters, and the greatest ensemble of voice over artists ever assembled for a daytime cartoon show.
      dtype=object)
```

Output 1-3

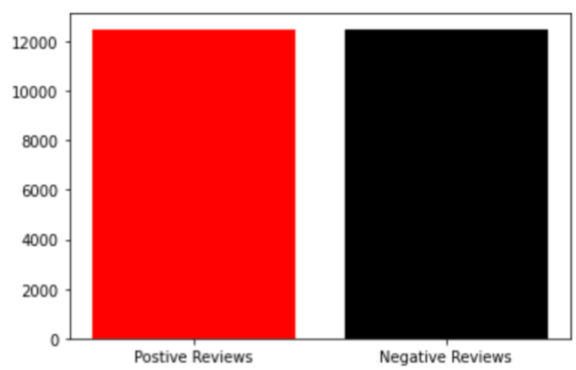
I then tried to create a bar plot to see how many positive and negative reviews there were in both datasets. What I found that there were equal number of negative and positive reviews in both. I also saw there were equal number in total between all the dataset splits shown in bar plots 1-4 through 1-6.



Shows number of Positive and Negative Reviews in total 1-4



# of pos and neg reviews in train data 1-5



# of pos and neg reviews in test data 1-6

Before creating the Neural Networks for this training, I had to make sure inputs of text had were of the same size, so I had to do padding of 64 words to get shorter words to the same size of the longer texts that included 64 words. The algorithm adds 0s on the end of these shorter encoded texts for padding [1].

Review research design and modeling methods

In this analysis I used a type of Deep Learning method which is called Recurrent Neural Networks. This type of Neural Network is used for sequential data [2]. The thing that is special about Recurrent Neurons is that it receives previous data with new data as shown in Figure 1-7 from the Geron book where x(t) represents present data input and y(t-1) represents previous days output [2]. With x(t) and y(t-1) we can create a formula represented by formula 1-8 which is a linear combination of x(t) and y(t-1) , and this creates a new output for a present day's neuron [2].

In the case of the analysis of NLP used in this analysis it is necessary to have a Bidirectional Layers as they look in both directions in both past and future to predict a target variable. In the case of text, it needs to look ahead to figure out what the combination of words builds up to, so the model can properly be encoded. There is also an embedding layer which transforms the encoded words that are put into vectors so that the models can train on them [2]. These vectors represent similar words or category of words. This particularly important for NLP.

Two models I am going to build is simple RNN and LSTM. LSTM is different to RNN in that it converges faster than RNN. Another thing about LSTM it has different types of gates for its neurons, for example, it has input gates which controls what are important inputs that the neuron can take in, it also has forget gates which is used to preserve long term information and erase information that is not needed, and it has output gates which distinguishes information that it should read and output [2]. In summary this is why the Long Short-Term Memory Neural Network got its name.

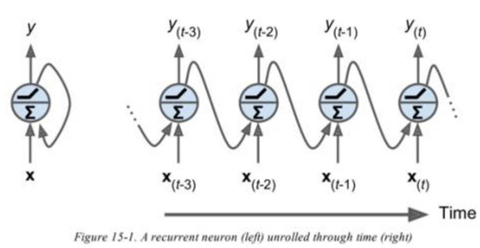


Figure 1-7 from Gerons

Equation 15-1. Output of a recurrent layer for a single instance

$$\mathbf{y}_{(t)} = \phi\left(\mathbf{W}_x{}^{\mathsf{T}}\mathbf{x}_{(t)} + \mathbf{W}_y{}^{\mathsf{T}}\mathbf{y}_{(t-1)} + \mathbf{b}\right)$$

Formula 1-8

For my Simple RNN, my layers are one Embedding Layer, one Bidirectional Layer, one Dense Layer all with 64 neurons, and one output layer with the sigmoid function which is used for Binary Classification. For LSTM model there is one Embedding Layer, Two Bidirectional Layers, one Dense Layer all with 64 neurons, one Dropout Layer for regularization, and one output layer with the sigmoid function used for Binary Classification. These are represented in 1-9, and 1-10. As you can see LSTM and Simple RNN are specified in the Bidirectional Layers which is the special parts of a RNN.

```
[22] model = tf.keras.Sequential([
         tf.keras.layers.Embedding(encoder.vocab_size, 64),
         tf.keras.layers.Bidirectional(tf.keras.layers.SimpleRNN(64)),
         tf.keras.layers.Dense(64, activation='relu'),
         tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
     ])
```

Simple RNN Model 1-9

```
model2 = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])
```

LSTM RNN 1-10

Review Results and Evaluate Model

After implementing the models, it was time to review the performance for each of the models. For the Simple RNN in 1-9, I got an underfitting problem, with a Training Accuracy of 0.6970 on 10 epochs and Test Accuracy of 0.6420. For the Test Data I got the Confusion Matrix in 1-13. After I found the confusion matrix, I computed the Precision and Recall scores as well as F1 Score and found Precision and F1 to be very low seen in 1-14 through 1-15. The F1 score is was 0.559 and precision score was the highest at 0.72, but recall score was the lowest around 0.454. I was unsatisfied with Simple RNN for the most part.

F1 score takes into account both precision and recall where 0 is the worst score and 1 is the best score [2]. Precision is a fraction of correctly identified positive results over all the instances identified as positive, while Recall is a fraction of correctly identified positive result over all the real positives. For Recall and Precision, 0 is bad, and 1 is the best score, so one could see they were all very good scores but could be unsatisfied with the Recall score.

In the LSTM model I got a pretty good Training Accuracy at 0.8386 and Test Set was around the same accuracy at 0.8224 which means this model was just about right. I then looked at the Confusion Matrix, F1 Score, Recall and Precision to see if they were underperforming as well, and I saw that they were all performing well compared to Simple RNN. Precision score was around 0.83, Recall score around 0.812 and F1 score around 0.82 all better than the scores for the Simple RNN model.

```
Epoch 1/10
391/391 [==============================] - 390s 998ms/step - loss: 0.6951 - accuracy: 0.4999 - val_loss: 0.6930 - val_accuracy: 0.4917
Epoch 2/10
391/391 [==============================] - 392s 1s/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6930 - val_accuracy: 0.4917
Epoch 3/10
391/391 [==============================] - 391s 1s/step - loss: 0.6930 - accuracy: 0.5000 - val_loss: 0.6925 - val_accuracy: 0.4917
Epoch 4/10
391/391 [==============================] - 393s 1s/step - loss: 0.6853 - accuracy: 0.5266 - val_loss: 0.6930 - val_accuracy: 0.4922
Epoch 5/10
391/391 [==============================] - 409s 1s/step - loss: 0.6909 - accuracy: 0.5024 - val_loss: 0.6857 - val_accuracy: 0.4932
Epoch 6/10
391/391 [==============================] - 408s 1s/step - loss: 0.6739 - accuracy: 0.5827 - val_loss: 0.6350 - val_accuracy: 0.7349
Epoch 7/10
391/391 [==============================] - 391s 1s/step - loss: 0.6249 - accuracy: 0.7105 - val_loss: 0.6915 - val_accuracy: 0.4938
Epoch 8/10
391/391 [==============================] - 396s 1s/step - loss: 0.6140 - accuracy: 0.7382 - val_loss: 0.6918 - val_accuracy: 0.4943
Epoch 9/10
391/391 [==============================] - 387s 990ms/step - loss: 0.6782 - accuracy: 0.5439 - val_loss: 0.6549 - val_accuracy: 0.6401
Epoch 10/10
391/391 [==============================] - 388s 993ms/step - loss: 0.6292 - accuracy: 0.6970 - val_loss: 0.6532 - val_accuracy: 0.6521
```

Epochs for Simple RNN 1-11

```
391/391 [==============================] - 74s 190ms/step - loss: 0.6596 - accuracy: 0.6430
Test Loss: 0.6595955491065979
Test Accuracy: 0.6429600119590759
```

Test Set Accuracy for RNN 1-12

```
test matrix
[[10398  2102]
 [ 6824  5676]]
```

Test Data Confusion Matrix for Simple RNN 1-13

```
precision test score:

0.7297505785548984

recall test score:
0.45408
```

Recall and Precision score RNN 1-14

```
f1 test score
0.5598185225367394
```

F1 Score RNN 1-15

```
Epoch 1/5
391/391 [==============================] - 1500s 4s/step - loss: 0.6989 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.4917
Epoch 2/5
391/391 [==============================] - 1528s 4s/step - loss: 0.6929 - accuracy: 0.5000 - val_loss: 0.6914 - val_accuracy: 0.4917
Epoch 3/5
391/391 [==============================] - 1553s 4s/step - loss: 0.6235 - accuracy: 0.7274 - val_loss: 0.5881 - val_accuracy: 0.7875
Epoch 4/5
391/391 [==============================] - 1652s 4s/step - loss: 0.5848 - accuracy: 0.8196 - val_loss: 0.5888 - val_accuracy: 0.7932
Epoch 5/5
391/391 [==============================] - 1693s 4s/step - loss: 0.5752 - accuracy: 0.8386 - val_loss: 0.5850 - val_accuracy: 0.8292
```

Epochs for LSTM 1-16

```
rint('Test Loss: {}'.format(test_loss))

391/391 [==============================] - 322s 823ms/step - loss: 0.5906 - accuracy: 0.8224
Test Accuracy: 0.8224400281906128
Test Loss: 0.5905635356903076
```

Test Set Accuracy LSTM 1-17

```
test matrix
[[10416  2084]
 [ 2355 10145]]
```

Test Data Confusion Matrix for LSTM 1-18

```
precision test score:
0.8295854117262246

recall test score:
0.8116
```

Recall and Precision score LSTM 1-19

```
1_score(ytruetr, clas
```

```
f1 score
0.8204941566581747
```

F1 score LSTM 1-20

Implementation and Programming

For implementation of the code, the first step is to import the packages as seen in code 1-21. The most important packages that I used below are the **Tensorflow** package, **Keras** package, **numpy** package and **matplotlib**. I also checked the versions of Tensorflow and Keras which were 2.3.0 and 2.4.0 respectively. The first major step was to load the IMBD dataset with 8,000 words. This was done with the function **tfds.load() from** Tensorflow package, using the path to dataset. It was easy to split up the dataset by using the load function. I split it up as there was a dictionary in the dataset which had the keys train and test set. I also had metadata on the file which was saved in the variable info.

Next I wanted to see how the encoder provided worked. This was done by saving the encoder from the metadata in a variable named 'encoder' as written by this line **encoder =**

info.features[‘text’].encoder. I also saw how many words the encoder could encode which was 8,185. Before implementing the models, I needed to pad to the longest text which was 64 words. Padding is useful so that it is ready for training, and all texts is of the same size. This was done by **.padded_batch().**

After adding padding, I then created a few barplots to see if negative and positive reviews were equal, and my plots confirmed they were. This was done by plt.bar() function from matplotlib package. After creating an EDA, I then implemented the model using keras and tensorflow package. I created the first model as seen in 1-9 and second model in 1-10. The model was created using **tf.keras.sequential() with Embedding() layer and Bidirectional() layers SimpleRNN or LMST explicitly stated.** I then used .compile to compile both models, and I used **.fit()** function to train the model which showed all the epochs and accuracy at each epochs between training and validation set.

The accuracy and loss were shown. I then got the accuracy of test data using **evaluate()** function. I then plotted the losses and training accuracy for both models using a custom made function which utilized Matplotlib. I then created confusion matrix using **SKLearn** package. I first used **.predict()** on test dataset, and then utilized **numpy.where** function to get the classes. I used **tensflow.concat** to get the true classes at axis =0 of tensor dataset object and saved true classes and predicted classes in a variable and passed it into confusion matrix function. Later I also imported **F1score, Precision, and Recall score functions** from **SKLearn** package as well. See in appendix below for more details.

```python
import datetime
from packaging import version
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

from collections import Counter
import numpy as np
import pandas as pd

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow import keras
from tensorflow.keras import models
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN,LSTM
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from tensorflow.keras.layers import Dropout, Flatten, Input, Dense
import tensorflow_datasets as tfds
#from plot_keras_history import plot_history
```

Code Cell 1-21

Exposition, problem description, Management recommendation

As examined above, I made a Simple RNN model and a LTSM Model. I then examined the results and I found that the scores were higher for the LTSM Model. I thought LTSM model would perform better as I realized it converges better to predictions. In my LTSM Model I got a Test Accuracy of 0.8224, a Precision Score of 0.83, a Recall Score of 0.812, and a F1 Score of 0.82. These were all relatively high. Therefore, **for my recommendation to management the LTSM Model should be chosen with one Embedding Layer, Two Bidirectional Layers, one Dense Layer all with 64 neurons, one Dropout Layer for regularization, and one output layer with the sigmoid function used for Binary Classification** as seen in model 1-10. 1-16 through 1-20 show the results of my model and justify my recommendation as indicated here.

For future analysis, I would like to figure out how make Simple RNN run better as I thought the scores were very low. I also want to figure out how I can make the LTSM accuracy go into high 90's along with Precision, Recall and F1 Scores. I think I would also want to play around with making the model fit faster as GPU/TPU took over 2 hours for both of these models to run. I used Google Colab for this analysis, but it still ended up taking forever.

References

[1] Srinivasan, S. (2020b, November 8th). *Sync Session 8* [Slides]. Canvas.

https://canvas.northwestern.edu/courses/125893/modules/items/1698004

[2] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2nd ed.). O'Reilly Media.

Appendix Next Page:

APPENDIX

```
import datetime
from packaging import version
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns

from collections import Counter
import numpy as np
import pandas as pd

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow import keras
from tensorflow.keras import models
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN,LSTM
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from tensorflow.keras.layers import Dropout, Flatten, Input, Dense
import tensorflow_datasets as tfds
#from plot_keras_history import plot_history
```

```
%matplotlib inline
np.set_printoptions(precision=3, suppress=True)
```

```
print("This notebook requires TensorFlow 2.0 or above")
print("TensorFlow version: ", tf.__version__)
assert version.parse(tf.__version__).release[0] >=2
```

```
This notebook requires TensorFlow 2.0 or above
TensorFlow version:  2.3.0
```

```
print("Keras version: ", keras.__version__)
```

```
Keras version:  2.4.0
```

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
def plot_graphs(history, metric):
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric], '')
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])
  plt.show()
```

```
dataset, info = tfds.load('imdb_reviews/subwords8k', with_info=True,
                          as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

```
WARNING:absl:TFDS datasets with text encoding are deprecated and will be removed in a future version. Instead, you should use the plain text version and tokenize the text using `tensorfl
Downloading and preparing dataset imdb_reviews/subwords8k/1.0.0 (download: 80.23 MiB, generated: Unknown size, total: 80.23 MiB) to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.
Dl Completed...: 100%                1/1 [3:43:38<00:00, 13418.77s/ url]
Dl Size...: 100%                     80/80 [3:43:38<00:00, 167.73s/ MiB]

Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0.incompleteTHO4MM/imdb_reviews-train.tfrecord
27%                                  6628/25000 [00:00<00:00, 66278.96 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0.incompleteTHO4MM/imdb_reviews-test.tfrecord
37%                                  9191/25000 [00:00<00:00, 91909.65 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0.incompleteTHO4MM/imdb_reviews-unsupervised.tfrecord
63%                                  31435/50000 [00:00<25:39, 12.06 examples/s]
WARNING:absl:Dataset is using deprecated text encoder API which will be removed soon. Please use the plain_text version of the dataset and migrate to `tensorflow_text`.
Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0. Subsequent calls will reuse this data.
```

```
imdb_reviews/plain_text
```

```
tfds.core.DatasetInfo(
    name='imdb_reviews',
    version=1.0.0,
    description='Large Movie Review Dataset.
This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for train
    homepage='http://ai.stanford.edu/~amaas/data/sentiment/',
    features=FeaturesDict({
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=2),
        'text': Text(shape=(None,), dtype=tf.int64, encoder=<SubwordTextEncoder vocab_size=8185>),
    }),
```

```
        total_num_examples=100000,
        splits={
            'test': 25000,
            'train': 25000,
            'unsupervised': 50000,
        },
        supervised_keys=('text', 'label'),
        citation="""@InProceedings{maas-EtAl:2011:ACL-HLT2011,
          author    = {Maas, Andrew L.  and  Daly, Raymond E.  and  Pham, Peter T.  and  Huang, Dan  and  Ng, Andrew Y.  and  Potts, Christopher},
          title     = {Learning Word Vectors for Sentiment Analysis},
          booktitle = {Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies},
          month     = {June},
          year      = {2011},
          address   = {Portland, Oregon, USA},
          publisher = {Association for Computational Linguistics},
          pages     = {142--150},
          url       = {http://www.aclweb.org/anthology/P11-1015}
        }""",
        redistribution_info=,
    )
```

```
dataset, info = tfds.load('imdb_reviews/plain_text', with_info=True,
                          as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']

texttr = tf.concat([x for x, y in train_dataset], axis = 0)
text = tf.concat([x  for x, y in test_dataset], axis = 0)

texttr.numpy()
text.numpy()
```

```
guez, EL MARIACHI. Add to that list Onur Tukel's absolutely amazing DING-A-LING-LESS. Flawless film-making, and as assured and as professional as any of the aforementioned movies. I haver
to tell a deeply humanist fable with a minimum of fuss. As an output from his Mexican era of film making, it was an invaluable talent to possess, with little money and extremely tight sch
e" the eleventh in a series that single handily ruined the parody genre. Now I\'ll admit it I have a soft spot for classics such as Airplane and The Naked Gun but you know you\'ve milked

one. Acting is OK at best, there's no action really and there is definitely no thrills. Capable actors with terrible script i think it could have been written better by a 10th grader. Fel
election. I personally recommend this. It's kind of the movie that can be seen by whole family at the same time without anyone feeling uncomfortable or getting bored. This cute movie will
ists ever assembled for a daytime cartoon show. This still remains as one of the highest rated daytime cartoon shows, and one of the most honored, winning several Emmy Awards."],
```

```
print('Vocabulary size: {}'.format(encoder.vocab_size))
```

```
Vocabulary size: 8185
```

```
sample_string = 'Hello Northwestern Data Science Students.'

encoded_string = encoder.encode(sample_string)
print('Encoded string is {}'.format(encoded_string))

original_string = encoder.decode(encoded_string)
print('The original string: "{}"'.format(original_string))
```

```
Encoded string is [4025, 222, 4277, 4413, 878, 1848, 2675, 2975, 2509, 6623, 8044, 7975]
The original string: "Hello Northwestern Data Science Students."
```

```
assert original_string == sample_string
```

```
for index in encoded_string:
  print('{} ----> {}'.format(index, encoder.decode([index])))
```

```
4025 ----> Hell
222 ----> o
4277 ----> North
4413 ----> western
878 ----> Da
1848 ----> ta
2675 ----> Sci
2975 ----> ence
2509 ----> Stu
6623 ----> dent
8044 ----> s
7975 ----> .
```


```
for index in encoded_string:
  print('{} ----> {}'.format(index, encoder.decode([index])))
```

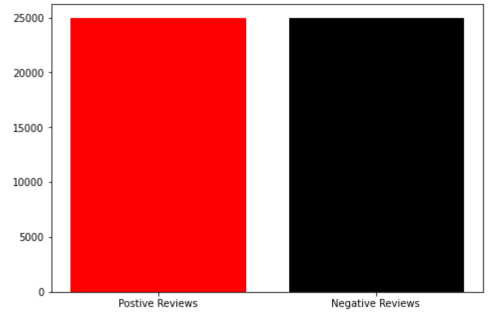
```
4025 ----> Hell
222 ----> o
4277 ----> North
4413 ----> western
878 ----> Da
1848 ----> ta
2675 ----> Sci
2975 ----> ence
2509 ----> Stu
6623 ----> dent
8044 ----> s
7975 ----> .
```


```
BUFFER_SIZE = 10000
BATCH_SIZE = 64
```

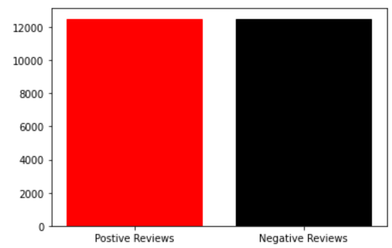
```
train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.padded_batch(BATCH_SIZE)

test_dataset = test_dataset.padded_batch(BATCH_SIZE)
```

```
ytruetr = tf.concat([y for x, y in train_dataset], axis = 0)
ytrue = tf.concat([y  for x, y in test_dataset], axis = 0)
```

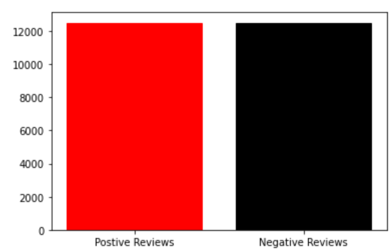
```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ytrue  = ytrue.numpy()
ytruetr = ytruetr.numpy()
negativesums = (ytruetr == 0).sum() + (ytrue == 0).sum()
positivesums = (ytruetr == 1).sum() + (ytrue == 1).sum()
bars = plt.bar(['Postive Reviews', 'Negative Reviews'], [positivesums, negativesums])
bars[0].set_color('r')
bars[1].set_color('black')
```




```
negativesums = (ytruetr == 0).sum()
positivesums = (ytruetr == 1).sum()
bars = plt.bar(['Postive Reviews', 'Negative Reviews'], [positivesums, negativesums])
bars[0].set_color('r')
bars[1].set_color('black')
```




```
negativesums = (ytrue == 0).sum()
positivesums = (ytrue == 1).sum()
bars = plt.bar(['Postive Reviews', 'Negative Reviews'], [positivesums, negativesums])
bars[0].set_color('r')
bars[1].set_color('black')
```



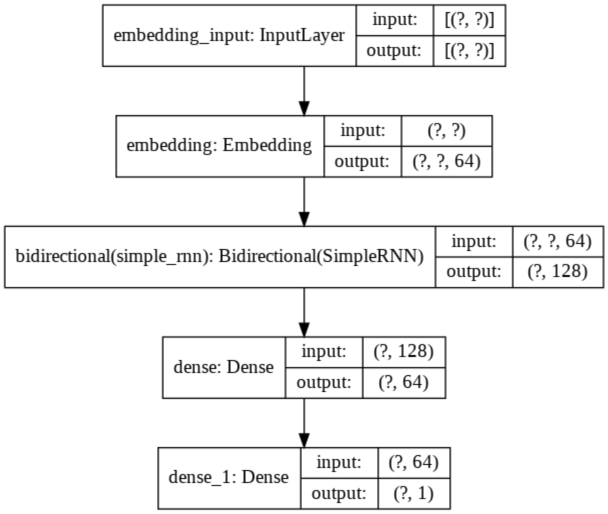
MODEL 1

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.SimpleRNN(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 64)          523840
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               16512
_________________________________________________________________
dense (Dense)                (None, 64)                8256
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 548,673
Trainable params: 548,673
Non-trainable params: 0
_________________________________________________________________
```

```
keras.utils.plot_model(model, "BinaryClassificationModel.png", show_shapes=True)
```




```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

```
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30)
```

```
Epoch 1/10
391/391 [==============================] - 390s 998ms/step - loss: 0.6951 - accuracy: 0.4999 - val_loss: 0.6930 - val_accuracy: 0.4917
Epoch 2/10
391/391 [==============================] - 392s 1s/step - loss: 0.6931 - accuracy: 0.5000 - val_loss: 0.6930 - val_accuracy: 0.4917
Epoch 3/10
391/391 [==============================] - 391s 1s/step - loss: 0.6930 - accuracy: 0.5000 - val_loss: 0.6925 - val_accuracy: 0.4917
Epoch 4/10
391/391 [==============================] - 393s 1s/step - loss: 0.6853 - accuracy: 0.5266 - val_loss: 0.6930 - val_accuracy: 0.4922
Epoch 5/10
391/391 [==============================] - 409s 1s/step - loss: 0.6909 - accuracy: 0.5024 - val_loss: 0.6857 - val_accuracy: 0.4932
Epoch 6/10
391/391 [==============================] - 408s 1s/step - loss: 0.6739 - accuracy: 0.5827 - val_loss: 0.6350 - val_accuracy: 0.7349
Epoch 7/10
391/391 [==============================] - 391s 1s/step - loss: 0.6249 - accuracy: 0.7105 - val_loss: 0.6915 - val_accuracy: 0.4938
Epoch 8/10
391/391 [==============================] - 396s 1s/step - loss: 0.6140 - accuracy: 0.7382 - val_loss: 0.6918 - val_accuracy: 0.4943
Epoch 9/10
391/391 [==============================] - 387s 990ms/step - loss: 0.6782 - accuracy: 0.5439 - val_loss: 0.6549 - val_accuracy: 0.6401
Epoch 10/10
391/391 [==============================] - 388s 993ms/step - loss: 0.6292 - accuracy: 0.6970 - val_loss: 0.6532 - val_accuracy: 0.6521
```

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
391/391 [==============================] - 74s 190ms/step - loss: 0.6596 - accuracy: 0.6430
Test Loss: 0.6595955491065979
Test Accuracy: 0.6429600119590759
```

```
def pad_to_size(vec, size):
  zeros = [0] * (size - len(vec))
  vec.extend(zeros)
  return vec
```

```
def sample_predict(sample_pred_text, model, pad):
  encoded_sample_pred_text = encoder.encode(sample_pred_text)

  if pad:
    encoded_sample_pred_text = pad_to_size(encoded_sample_pred_text, 64)
  encoded_sample_pred_text = tf.cast(encoded_sample_pred_text, tf.float32)
  predictions = model.predict(tf.expand_dims(encoded_sample_pred_text, 0))

  return (predictions)
```

```
sample_pred_text = ('The movie was cool. The animation and the graphics '
                    'were out of this world. I would recommend this movie.')
predictions = sample_predict(sample_pred_text, model ,pad=False)
print(predictions)
```

```
[[0.992]]
```

```
sample_pred_text = ('The movie was cool. The animation and the graphics '
                    'were out of this world. I would recommend this movie.')
predictions = sample_predict(sample_pred_text,model,pad=True)
print(predictions)
```

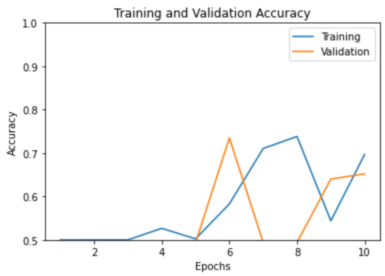
```
[[0.053]]
```

```
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
plt.plot(range(1, len(acc) + 1), history.history['accuracy'], label = 'Training')
plt.plot(range(1, len(val_acc) + 1), history.history['val_accuracy'], label = 'Validation')
plt.ylim([0.5, 1.0])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
plt.plot(range(1, len(loss) + 1), history.history['loss'], label = 'Training')
plt.plot(range(1, len(val_loss) + 1), history.history['val_loss'], label = 'Validation')
plt.ylim([0.0, 1.0])
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```


```
from sklearn.metrics import confusion_matrix
predictions1 = model.predict(test_dataset)
predictionstr1 = model.predict(train_dataset)
classes = np.where(predictions1 >= 0.5, 1, 0)
classestr = np.where(predictionstr1 >= 0.5, 1, 0)
ytruetr = tf.concat([y for x, y in train_dataset], axis = 0)
ytrue = tf.concat([y for x, y in test_dataset], axis = 0)
print("test matrix")
print(confusion_matrix(ytrue,classes))
print(" ")
print("train matrix")
print(confusion_matrix(ytruetr,classestr))
```

```
test matrix
[[10398  2102]
 [ 6824  5676]]

train matrix
[[8330 4170]
 [8437 4063]]
```

```
from sklearn.metrics import precision_score, recall_score
print("precision test score: ")
print(precision_score(ytrue,classes))
print(" ")
print("recall test score: ")
print(recall_score(ytrue,classes))

print("precision train score: ")
print(precision_score(ytruetr,classestr))
print(" ")
print("recall train score: ")
print(recall_score(ytruetr,classestr))
```

```
precision test score:
```

```
0.7297505785548984

recall test score:
0.45408
precision train score:
0.4935017612049071

recall train score:
0.32504
```

```
from sklearn.metrics import f1_score
print("f1 test score")
print(f1_score(ytrue,classes))
print(" ")
print("f1 train score")
f1_score(ytruetr,classestr)
```

```
f1 test score
0.5598185225367394

f1 train score
0.39193556166497856
```

MODEL 2

```
model2 = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])
```

```
model2.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

```
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
bidirectional_1 (Bidirection (None, None, 128)         66048
_________________________________________________________________
bidirectional_2 (Bidirection (None, 64)                41216
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 635,329
Trainable params: 635,329
Non-trainable params: 0
_________________________________________________________________
```

```
keras.utils.plot_model(model2, "LSTMBinaryClassificationModel.png", show_shapes=True)
```

```
history2 = model2.fit(train_dataset, epochs=5,
                    validation_data=test_dataset,
                    validation_steps=30)
```

```
Epoch 1/5
391/391 [==============================] - 1500s 4s/step - loss: 0.6989 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.4917
Epoch 2/5
391/391 [==============================] - 1528s 4s/step - loss: 0.6929 - accuracy: 0.5000 - val_loss: 0.6914 - val_accuracy: 0.4917
Epoch 3/5
391/391 [==============================] - 1553s 4s/step - loss: 0.6235 - accuracy: 0.7274 - val_loss: 0.5881 - val_accuracy: 0.7875
Epoch 4/5
391/391 [==============================] - 1652s 4s/step - loss: 0.5848 - accuracy: 0.8196 - val_loss: 0.5888 - val_accuracy: 0.7932
Epoch 5/5
391/391 [==============================] - 1693s 4s/step - loss: 0.5752 - accuracy: 0.8386 - val_loss: 0.5850 - val_accuracy: 0.8292
```

```
test_loss, test_acc = model2.evaluate(test_dataset)
print('Test Accuracy: {}'.format(test_acc))
print('Test Loss: {}'.format(test_loss))
```

```
391/391 [==============================] - 322s 823ms/step - loss: 0.5906 - accuracy: 0.8224
Test Accuracy: 0.8224400281906128
Test Loss: 0.5905635356903076
```

```
sample_pred_text = ('The movie was not good. The animation and the graphics '
                    'were terrible. I would not recommend this movie.')
predictions = sample_predict(sample_pred_text,model, pad=False)
print(predictions)
```

```
[[0.308]]
```

```
sample_pred_text = ('The movie was not good. The animation and the graphics '
                    'were terrible. I would not recommend this movie.')
predictions = sample_predict(sample_pred_text, model, pad=True)
print(predictions)
```

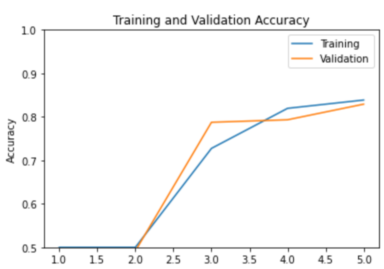
```
[[0.007]]
```

```
history_dict2 = history2.history
history_dict2.keys()
```

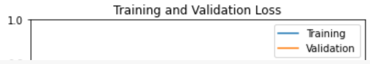
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
acc = history2.history['accuracy']
val_acc = history2.history['val_accuracy']
loss = history2.history['loss']
val_loss = history2.history['val_loss']
```

```
plt.plot(range(1, len(acc) + 1), history2.history['accuracy'], label = 'Training')
plt.plot(range(1, len(val_acc) + 1), history2.history['val_accuracy'], label = 'Validation')
plt.ylim([0.5, 1.0])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
plt.plot(range(1, len(loss) + 1), history2.history['loss'], label = 'Training')
plt.plot(range(1, len(val_loss) + 1), history2.history['val_loss'], label = 'Validation')
plt.ylim([0.0, 1.0])
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
predictionstr2 = model2.predict(train_dataset)
predictions2 = model2.predict(test_dataset)
classestr2 = np.where(predictionstr2 >= 0.5, 1, 0)
classes2 = np.where(predictions2 >= 0.5, 1, 0)
print('test matrix')
print(confusion_matrix(ytrue, classes2))
print(" ")
print("train matrix")
print(confusion_matrix(ytruetr, classestr2))
```

```
test matrix
[[10416  2084]
 [ 2355 10145]]

train matrix
[[6491 6009]
 [6489 6011]]
```

```
print("precision test score: ")
print(precision_score(ytrue, classes2))
print(" ")
print("recall test score: ")
print(recall_score(ytrue, classes2))

print("precision train score: ")
print(precision_score(ytruetr, classestr2))
print(" ")
print("recall train score: ")
recall_score(ytruetr, classestr2)
```

```
precision test score:
0.8295854117262246

recall test score:
0.8116
precision train score:
0.5000831946755407

recall train score:
0.48088
```

```
print("f1 score")
print(f1_score(ytrue, classes2))
print(" ")
print("f1 train score")
f1_score(ytruetr, classestr2)
```

```
f1 score
0.8204941566581747

f1 train score
0.49029363784665575
```