Modeling_NN_GloVe_LSTM

July 27, 2022

In this notebook, we will use GloVe with LSTM.

1 Importing Libraries

[ ]:
```
import numpy as np
import pandas as pd
import string


import seaborn as sns
import matplotlib.pyplot as plt


from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, plot_confusion_matrix
from sklearn.preprocessing import LabelEncoder


from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import ColumnTransformer
from gensim.models import word2vec

import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Flatten
from tensorflow.keras.layers import Dropout, Activation, Bidirectional,␣
↪GlobalMaxPool1D
from tensorflow.keras.models import Sequential
from tensorflow.keras import initializers, regularizers, constraints,␣
↪optimizers, layers
from tensorflow.keras.preprocessing import text, sequence
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.callbacks import EarlyStopping

import tensorflow_hub as hub
import tensorflow_datasets as tfds
```

```
from  IPython import display

import pathlib
import shutil
import tempfile

!pip install -q git+https://github.com/tensorflow/docs

import tensorflow_docs as tfdocs
import tensorflow_docs.modeling
import tensorflow_docs.plots

print("Version: ", tf.__version__)
print("Hub version: ", hub.__version__)
print("GPU is", "available" if tf.config.list_physical_devices('GPU') else "NOT␣
 ↪AVAILABLE")

logdir = pathlib.Path(tempfile.mkdtemp())/"tensorboard_logs"
shutil.rmtree(logdir, ignore_errors=True)


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

```
Version:  2.8.2
Hub version:  0.12.0
GPU is available
```

[ ]:
```
!nvidia-smi
```

```
Sat Jul  9 10:16:20 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   49C    P8    10W /  70W |      3MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
```

```
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

2 Functions We Use

The functionsthat we use in this notebook are located in this section. The first function is the neural network that we will train on the training set. The next two functions perform some basic cleaning and the last function prints the result of the training.

2.0.1 Cleaning the data

```
[ ]: def new_cleaning(data):
         cleaned_data = data.replace("[", "").replace("]", "").replace("'", "")\
                                         .replace(" ", "").split(",")

         cleaned_sentence = " ".join(cleaned_data)


         return cleaned_sentence
```

2.0.2 Printing the results

```
[ ]: def print_results(model):



         train_loss = model.history.history["loss"]
         train_acc  = model.history.history["accuracy"]

         test_loss = model.history.history["val_loss"]
         test_acc  = model.history.history["val_accuracy"]


         sns.set(font_scale=1)

         cf_matrix_test  = confusion_matrix(y_test, np.rint(model.
      ↪predict(padded_test)))
         cf_matrix_train = confusion_matrix(y_train, np.rint(model.
      ↪predict(padded_train)))


         fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10,10))
         ax1 = axes[0][0]

         g = sns.heatmap(cf_matrix_test, annot=True, cmap='Blues', ax = ax1)
```

```
g.xaxis.set_ticklabels(['Fake','True'])
g.yaxis.set_ticklabels(['Fake','True'])


ax1.set_title("Confusion Matrix for Test Set")

ax2 = axes[0][1]
g = sns.heatmap(cf_matrix_train, annot=True, cmap='Blues', ax = ax2)
g.xaxis.set_ticklabels(['Fake','True'])
g.yaxis.set_ticklabels(['Fake','True'])


ax2.set_title("Confusion Matrix for Train Set")



ax3 = axes[1][0]

g = sns.lineplot(x = range(1,len(train_loss)+1),
                 y = train_loss,
                 label = "train_loss", ax = ax3);
g = sns.lineplot(x = range(1,len(test_loss)+1),
                 y = test_loss,
                 label = "test_loss", ax = ax3);



ax3.set_title("Loss vs. Epochs")



ax4 = axes[1][1]

g = sns.lineplot(x = range(1,len(train_acc)+1),
                 y = train_acc,
                 label = "train_accuracy", ax = ax4);
g = sns.lineplot(x = range(1,len(test_acc)+1),
                 y = test_acc,
                 label = "test_accuracy", ax = ax4);


ax4.set_title("Accuracy vs. Epochs")


### Presenting Classification Report as a DataFrame
```

```
    train_class = classification_report(y_train, np.rint(model.
 ↪predict(padded_train)),
                                        output_dict = True)
    test_class  = classification_report(y_test, np.rint(model.
 ↪predict(padded_test)),
                                        output_dict = True)

    train_df = pd.DataFrame(train_class)
    test_df  = pd.DataFrame(test_class)

    train_df["data"] = "TRAIN"
    test_df["data"] = "TEST"


    report = pd.concat([test_df, train_df], axis = 0)
    report.rename(columns = {"1": f"{list(le.inverse_transform([1]))[0]}",
                             "0": f"{list(le.inverse_transform([0]))[0]}"},␣
 ↪inplace = True)
    report["index"] = list(report.index)

    report.set_index(["data", "index"], inplace = True)


    for item in list(report.columns):
        report[item] = report[item].apply(lambda x: np.round(x,2))

    return report
```

3 Importing Data

[ ]:
```
test = pd.read_csv("/content/drive/MyDrive/Fake-Real-News-Classification/
 ↪Modeling/train_test/test.csv")
train = pd.read_csv("/content/drive/MyDrive/Fake-Real-News-Classification/
 ↪Modeling/train_test/train.csv")



data  = [train, test]

for df in data:

    df.drop("Unnamed: 0", axis = 1, inplace = True)
    df["cleaned"] =  df["cleaned"].apply(lambda x: new_cleaning(x))

    df["cleaned_glove"] =  df["for_glove"].apply(lambda x: new_cleaning(x))
```

```
print("DONE!")
```

```
DONE!
```

[ ]:
```
le = LabelEncoder()

X_train = train["cleaned"].values
y_train = le.fit_transform(train["label"])

X_test = test["cleaned"].values
y_test = le.transform(test["label"])
```

[ ]:
```
X_train[0].split()
```

[ ]:
```
['april', 'giraffe', 'zoo', 'want', 'name', 'calf']
```

4 Importing GloVe

[ ]:
```
total_vocabulary = set(word for headline in X_train for word in headline.
 ↪split())
print(len(total_vocabulary))


glove = {}
with open('/content/drive/MyDrive/Fake-Real-News-Classification/Modeling/glove.
 ↪840B.300d.txt', 'rb') as f:
    for line in f:
        parts = line.split()
        word = parts[0].decode('utf-8')
        if word in total_vocabulary:
            vector = np.array(parts[1:], dtype=np.float32)
            glove[word] = vector



print("DONE!")
```

```
94186
DONE!
```

[ ]:
```
len(list(glove.keys()))
glove[list(glove.keys())[0]].shape[0]
```

[ ]:
```
300
```

[ ]:
```
max_features = 1000
# max_len = glove[list(glove.keys())[0]].shape[0]
max_len = 250

## Train Set
tokenizer = text.Tokenizer(num_words= max_features)#, oov_token= "<OOV>")


tokenizer.fit_on_texts(X_train)

word_index = tokenizer.word_index

tokenized_train = tokenizer.texts_to_sequences(X_train)
padded_train = pad_sequences(tokenized_train, maxlen = max_len,␣
↪truncating="post")


## Test Set
tokenized_test = tokenizer.texts_to_sequences(X_test)
padded_test    = pad_sequences(tokenized_test,maxlen=max_len)

print(len(word_index))
```

93782

[ ]:
```
embedding_dim = glove[list(glove.keys())[0]].shape[0]
embedding_matrix = np.zeros((len(word_index) + 1 , embedding_dim))

for key, val in word_index.items():

   if key in glove.keys():
      embedding_matrix[val] = glove[key]
```

[ ]:
```
embedding_layer = Embedding(embedding_matrix.shape[0], embedding_matrix.
↪shape[1],
                            weights = [embedding_matrix],
                            input_length = max_len,
                            trainable = False)
```

[ ]:
```
#Defining Neural Network
model = Sequential()
#Non-trainable embeddidng layer
model.add(Embedding(embedding_matrix.shape[0], embedding_matrix.shape[1],
                    weights=[embedding_matrix], input_length=max_len,␣
↪trainable=False))
#LSTM
```

```python
model.add(LSTM(units=64 , return_sequences = True , recurrent_dropout = 0.25 ,␣
↪dropout = 0.5))
model.add(LSTM(units=32 , recurrent_dropout = 0.1 , dropout = 0.5))
model.add(Flatten())
model.add(Dense(units = 40 , activation = 'relu'))
model.add(Dropout(0.5))
# model.add(Dense(units = 20 , activation = 'relu'))
# model.add(Dropout(0.5))
# model.add(Dense(units = 10 , activation = 'relu'))
# model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=tf.keras.optimizers.Adam(lr = 0.001),␣
↪loss='binary_crossentropy', metrics=['accuracy'])
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.
```

[ ]:
```python
model.fit(padded_train, y_train, epochs=40, batch_size=3000,
          validation_data=(padded_test, y_test))
print("fitting to the model is DONE!")
print_results(model)
```

```
Epoch 1/40
15/15 [==============================] - 39s 2s/step - loss: 0.6275 - accuracy:
0.6813 - val_loss: 0.5044 - val_accuracy: 0.7864
Epoch 2/40
15/15 [==============================] - 33s 2s/step - loss: 0.4936 - accuracy:
0.7893 - val_loss: 0.4319 - val_accuracy: 0.8145
Epoch 3/40
15/15 [==============================] - 32s 2s/step - loss: 0.4429 - accuracy:
0.8133 - val_loss: 0.4520 - val_accuracy: 0.8136
Epoch 4/40
15/15 [==============================] - 33s 2s/step - loss: 0.4065 - accuracy:
0.8320 - val_loss: 0.3824 - val_accuracy: 0.8333
Epoch 5/40
15/15 [==============================] - 33s 2s/step - loss: 0.4264 - accuracy:
0.8000 - val_loss: 0.3804 - val_accuracy: 0.8269
Epoch 6/40
15/15 [==============================] - 32s 2s/step - loss: 0.3787 - accuracy:
0.8287 - val_loss: 0.3872 - val_accuracy: 0.8177
Epoch 7/40
15/15 [==============================] - 32s 2s/step - loss: 0.3573 - accuracy:
0.8412 - val_loss: 0.3629 - val_accuracy: 0.8234
Epoch 8/40
15/15 [==============================] - 33s 2s/step - loss: 0.3505 - accuracy:
```

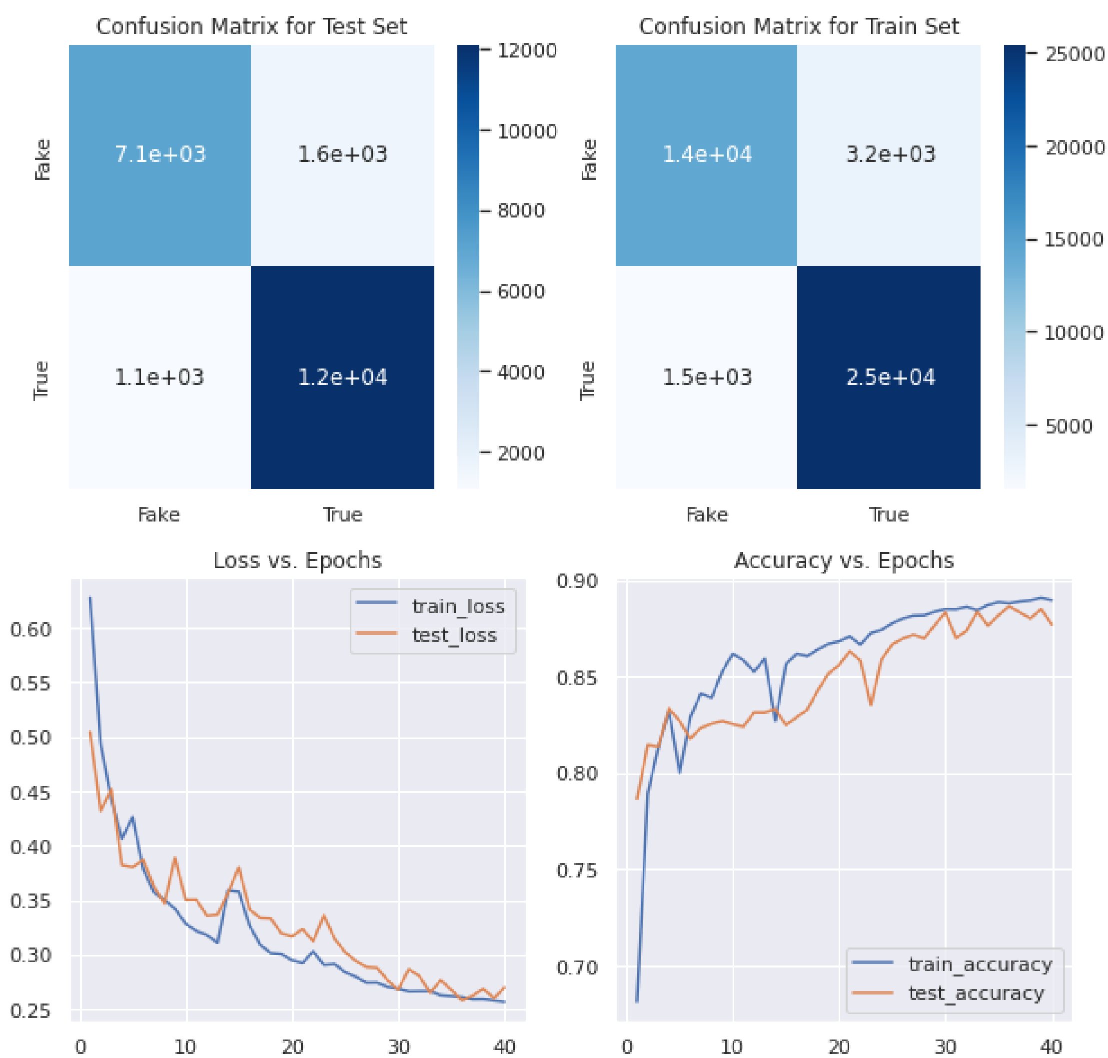
```
0.8389 - val_loss: 0.3469 - val_accuracy: 0.8255
Epoch 9/40
15/15 [==============================] - 34s 2s/step - loss: 0.3423 - accuracy:
0.8525 - val_loss: 0.3892 - val_accuracy: 0.8269
Epoch 10/40
15/15 [==============================] - 33s 2s/step - loss: 0.3287 - accuracy:
0.8618 - val_loss: 0.3505 - val_accuracy: 0.8253
Epoch 11/40
15/15 [==============================] - 32s 2s/step - loss: 0.3219 - accuracy:
0.8584 - val_loss: 0.3506 - val_accuracy: 0.8240
Epoch 12/40
15/15 [==============================] - 32s 2s/step - loss: 0.3182 - accuracy:
0.8525 - val_loss: 0.3359 - val_accuracy: 0.8314
Epoch 13/40
15/15 [==============================] - 33s 2s/step - loss: 0.3110 - accuracy:
0.8592 - val_loss: 0.3369 - val_accuracy: 0.8313
Epoch 14/40
15/15 [==============================] - 33s 2s/step - loss: 0.3592 - accuracy:
0.8270 - val_loss: 0.3571 - val_accuracy: 0.8330
Epoch 15/40
15/15 [==============================] - 33s 2s/step - loss: 0.3579 - accuracy:
0.8564 - val_loss: 0.3804 - val_accuracy: 0.8248
Epoch 16/40
15/15 [==============================] - 34s 2s/step - loss: 0.3271 - accuracy:
0.8618 - val_loss: 0.3417 - val_accuracy: 0.8289
Epoch 17/40
15/15 [==============================] - 33s 2s/step - loss: 0.3094 - accuracy:
0.8606 - val_loss: 0.3339 - val_accuracy: 0.8327
Epoch 18/40
15/15 [==============================] - 33s 2s/step - loss: 0.3016 - accuracy:
0.8640 - val_loss: 0.3334 - val_accuracy: 0.8428
Epoch 19/40
15/15 [==============================] - 34s 2s/step - loss: 0.3007 - accuracy:
0.8669 - val_loss: 0.3198 - val_accuracy: 0.8515
Epoch 20/40
15/15 [==============================] - 32s 2s/step - loss: 0.2952 - accuracy:
0.8682 - val_loss: 0.3170 - val_accuracy: 0.8559
Epoch 21/40
15/15 [==============================] - 33s 2s/step - loss: 0.2927 - accuracy:
0.8708 - val_loss: 0.3236 - val_accuracy: 0.8630
Epoch 22/40
15/15 [==============================] - 33s 2s/step - loss: 0.3032 - accuracy:
0.8665 - val_loss: 0.3126 - val_accuracy: 0.8583
Epoch 23/40
15/15 [==============================] - 33s 2s/step - loss: 0.2907 - accuracy:
0.8725 - val_loss: 0.3364 - val_accuracy: 0.8351
Epoch 24/40
15/15 [==============================] - 33s 2s/step - loss: 0.2917 - accuracy:
```

```
0.8742 - val_loss: 0.3150 - val_accuracy: 0.8589
Epoch 25/40
15/15 [==============================] - 33s 2s/step - loss: 0.2841 - accuracy:
0.8776 - val_loss: 0.3024 - val_accuracy: 0.8667
Epoch 26/40
15/15 [==============================] - 32s 2s/step - loss: 0.2800 - accuracy:
0.8800 - val_loss: 0.2945 - val_accuracy: 0.8697
Epoch 27/40
15/15 [==============================] - 33s 2s/step - loss: 0.2745 - accuracy:
0.8815 - val_loss: 0.2889 - val_accuracy: 0.8716
Epoch 28/40
15/15 [==============================] - 34s 2s/step - loss: 0.2746 - accuracy:
0.8817 - val_loss: 0.2881 - val_accuracy: 0.8698
Epoch 29/40
15/15 [==============================] - 33s 2s/step - loss: 0.2706 - accuracy:
0.8836 - val_loss: 0.2767 - val_accuracy: 0.8766
Epoch 30/40
15/15 [==============================] - 32s 2s/step - loss: 0.2691 - accuracy:
0.8850 - val_loss: 0.2681 - val_accuracy: 0.8834
Epoch 31/40
15/15 [==============================] - 32s 2s/step - loss: 0.2665 - accuracy:
0.8848 - val_loss: 0.2869 - val_accuracy: 0.8699
Epoch 32/40
15/15 [==============================] - 33s 2s/step - loss: 0.2667 - accuracy:
0.8861 - val_loss: 0.2806 - val_accuracy: 0.8737
Epoch 33/40
15/15 [==============================] - 33s 2s/step - loss: 0.2667 - accuracy:
0.8844 - val_loss: 0.2652 - val_accuracy: 0.8836
Epoch 34/40
15/15 [==============================] - 32s 2s/step - loss: 0.2630 - accuracy:
0.8871 - val_loss: 0.2768 - val_accuracy: 0.8763
Epoch 35/40
15/15 [==============================] - 33s 2s/step - loss: 0.2622 - accuracy:
0.8885 - val_loss: 0.2680 - val_accuracy: 0.8818
Epoch 36/40
15/15 [==============================] - 32s 2s/step - loss: 0.2613 - accuracy:
0.8879 - val_loss: 0.2583 - val_accuracy: 0.8864
Epoch 37/40
15/15 [==============================] - 32s 2s/step - loss: 0.2594 - accuracy:
0.8889 - val_loss: 0.2626 - val_accuracy: 0.8833
Epoch 38/40
15/15 [==============================] - 34s 2s/step - loss: 0.2595 - accuracy:
0.8895 - val_loss: 0.2690 - val_accuracy: 0.8801
Epoch 39/40
15/15 [==============================] - 33s 2s/step - loss: 0.2582 - accuracy:
0.8908 - val_loss: 0.2603 - val_accuracy: 0.8850
Epoch 40/40
15/15 [==============================] - 32s 2s/step - loss: 0.2570 - accuracy:
```

```
0.8896 - val_loss: 0.2702 - val_accuracy: 0.8770
fitting to the model is DONE!
```

[ ]:

data	index	Fake	True	accuracy	macro avg	weighted avg
TEST	precision	0.87	0.88	0.88	0.88	0.88
	recall	0.81	0.92	0.88	0.87	0.88
	f1-score	0.84	0.90	0.88	0.87	0.88
	support	8719.00	13194.00	0.88	21913.00	21913.00
TRAIN	precision	0.90	0.89	0.89	0.90	0.89
	recall	0.82	0.94	0.89	0.88	0.89
	f1-score	0.86	0.91	0.89	0.89	0.89
	support	17502.00	26987.00	0.89	44489.00	44489.00



[ ]:
```
import pickle
from joblib import dump, load
dump(model, '/content/drive/MyDrive/Fake-Real-News-Classification/
↪pickled_nn_models/model__nn_glove.joblib')
```

```
INFO:tensorflow:Assets written to:
ram://62078037-38f8-4f89-97de-9487083fad15/assets
```

```
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7efd89e21110> has the
same name 'LSTMCell' as a built-in Keras object. Consider renaming <class
'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading with
`tf.keras.models.load_model`. If renaming is not possible, pass the object in
the `custom_objects` parameter of the load function.
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7efd89e58b10> has the
same name 'LSTMCell' as a built-in Keras object. Consider renaming <class
'keras.layers.recurrent.LSTMCell'> to avoid naming conflicts when loading with
`tf.keras.models.load_model`. If renaming is not possible, pass the object in
the `custom_objects` parameter of the load function.
```

[ ]:
```
['/content/drive/MyDrive/Fake-Real-News-
Classification/pickled_nn_models/model__nn_glove.joblib']
```

[ ]:
```

```