Text Classification Assignment

In this assignment we were tasked with finding a data set and using Keras deep learning to predict things about it. We chose a data set of potential job postings with some of them being fake. Our machine was trained on this data to detect whether a job posting was real or not based on the job description.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models

from sklearn.preprocessing import LabelEncoder
import pickle
import numpy as np
import pandas as pd
import seaborn as sns
```

Distribution graph

This segment creates the dataframe from the given csv file and displays a graph showing the distribution of the target classes.

```
df = pd.read_csv('fake_job_postings.csv', engine='python', encoding='utf-8', error_bad_lines=False)
df.description=df.description.astype(str)
print(df.head())
```

```
/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshell.py:3326: FutureWarning: The error_bad_lines argument has been dep

  exec(code_obj, self.user_global_ns, self.user_ns)
   job_id                                      title            location  \
0       1                           Marketing Intern    US, NY, New York
1       2  Customer Service - Cloud Video Production      NZ, , Auckland
2       3    Commissioning Machinery Assistant (CMA)       US, IA, Wever
3       4          Account Executive - Washington DC  US, DC, Washington
4       5                        Bill Review Manager  US, FL, Fort Worth

  department salary_range                                    company_profile  \
0  Marketing          NaN  We're Food52, and we've created a groundbreaki...
1    Success          NaN  90 Seconds, the worlds Cloud Video Production ...
2        NaN          NaN  Valor Services provides Workforce Solutions th...
3      Sales          NaN  Our passion for improving quality of life thro...
4        NaN          NaN  SpotSource Solutions LLC is a Global Human Cap...

                                         description  \
0  Food52, a fast-growing, James Beard Award-winn...
1  Organised - Focused - Vibrant - Awesome!Do you...
2  Our client, located in Houston, is actively se...
3  THE COMPANY: ESRI – Environmental Systems Rese...
4  JOB TITLE: Itemization Review ManagerLOCATION:...

                                        requirements  \
0  Experience with content management systems a m...
1  What we expect from you:Your key responsibilit...
2  Implement pre-commissioning and commissioning ...
3  EDUCATION: Bachelor’s or Master’s in GIS, busi...
4  QUALIFICATIONS:RN license in the State of Texa...

                                            benefits  telecommuting  \
0                                                NaN              0
1  What you will get from usThrough being part of...              0
2                                                NaN              0
3  Our culture is anything but corporate—we have ...              0
4                              Full Benefits Offered              0

   has_company_logo  has_questions employment_type required_experience  \
0                 1              0           Other          Internship
1                 1              0       Full-time      Not Applicable
2                 1              0             NaN                 NaN
3                 1              0       Full-time    Mid-Senior level
4                 1              1       Full-time    Mid-Senior level

  required_education                   industry              function  \
0                NaN                        NaN             Marketing
```

```
1               NaN  Marketing and Advertising      Customer Service
2               NaN                        NaN                   NaN
3  Bachelor's Degree          Computer Software                 Sales
4  Bachelor's Degree     Hospital & Health Care  Health Care Provider

   fraudulent
0           0
1           0
```

Splitting our data into train and test sets

```
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
print("train data size: ", train.shape)
print("test data size: ", test.shape)
```

```
train data size:  (14292, 18)
test data size:  (3588, 18)
```

Sequential Model

This model is the first sequential model we tried which provided great results with an average 97% accuracy score

```
num_labels = 2
vocab_size = 25000
batch_size = 100

tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.description)

x_train = tokenizer.texts_to_matrix(train.description, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.description, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.fraudulent)
y_train = encoder.transform(train.fraudulent)
y_test = encoder.transform(test.fraudulent)

# check shape
print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
print("test first five labels:", y_test[:5])
```

```
train shapes: (14292, 25000) (14292,)
test shapes: (3588, 25000) (3588,)
test first five labels: [0 0 0 0 0]
```

```
# fit model
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal', activation='relu'))
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1)
```

```
Epoch 1/30
79/79 [==============================] - 3s 28ms/step - loss: 0.2349 - accuracy: 0.9434 - val_loss: 0.1423 - val_accuracy: 0.9736
Epoch 2/30
79/79 [==============================] - 2s 23ms/step - loss: 0.0892 - accuracy: 0.9718 - val_loss: 0.1335 - val_accuracy: 0.9782
Epoch 3/30
79/79 [==============================] - 2s 23ms/step - loss: 0.0407 - accuracy: 0.9895 - val_loss: 0.1450 - val_accuracy: 0.9782
```

```
Epoch 4/30
79/79 [==============================] - 2s 22ms/step - loss: 0.0186 - accuracy: 0.9962 - val_loss: 0.1696 - val_accuracy: 0.9771
Epoch 5/30
79/79 [==============================] - 2s 23ms/step - loss: 0.0103 - accuracy: 0.9978 - val_loss: 0.1887 - val_accuracy: 0.9794
Epoch 6/30
79/79 [==============================] - 2s 23ms/step - loss: 0.0067 - accuracy: 0.9990 - val_loss: 0.2135 - val_accuracy: 0.9782
Epoch 7/30
79/79 [==============================] - 2s 23ms/step - loss: 0.0052 - accuracy: 0.9990 - val_loss: 0.2279 - val_accuracy: 0.9782
Epoch 8/30
79/79 [==============================] - 2s 23ms/step - loss: 0.0038 - accuracy: 0.9994 - val_loss: 0.2417 - val_accuracy: 0.9805
Epoch 9/30
79/79 [==============================] - 2s 23ms/step - loss: 0.0034 - accuracy: 0.9994 - val_loss: 0.2517 - val_accuracy: 0.9794
Epoch 10/30
79/79 [==============================] - 2s 22ms/step - loss: 0.0032 - accuracy: 0.9992 - val_loss: 0.2522 - val_accuracy: 0.9782
Epoch 11/30
79/79 [==============================] - 2s 22ms/step - loss: 0.0027 - accuracy: 0.9992 - val_loss: 0.2671 - val_accuracy: 0.9794
Epoch 12/30
79/79 [==============================] - 2s 22ms/step - loss: 0.0021 - accuracy: 0.9994 - val_loss: 0.2811 - val_accuracy: 0.9805
Epoch 13/30
79/79 [==============================] - 2s 22ms/step - loss: 0.0019 - accuracy: 0.9995 - val_loss: 0.2780 - val_accuracy: 0.9782
Epoch 14/30
79/79 [==============================] - 2s 28ms/step - loss: 0.0016 - accuracy: 0.9996 - val_loss: 0.2898 - val_accuracy: 0.9805
Epoch 15/30
79/79 [==============================] - 2s 29ms/step - loss: 0.0019 - accuracy: 0.9995 - val_loss: 0.2983 - val_accuracy: 0.9805
Epoch 16/30
79/79 [==============================] - 2s 22ms/step - loss: 0.0016 - accuracy: 0.9996 - val_loss: 0.2997 - val_accuracy: 0.9805
Epoch 17/30
79/79 [==============================] - 2s 22ms/step - loss: 0.0011 - accuracy: 0.9997 - val_loss: 0.2989 - val_accuracy: 0.9794
Epoch 18/30
79/79 [==============================] - 2s 22ms/step - loss: 9.3982e-04 - accuracy: 0.9997 - val_loss: 0.3119 - val_accuracy: 0.9805
Epoch 19/30
79/79 [==============================] - 2s 22ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.3085 - val_accuracy: 0.9794
Epoch 20/30
79/79 [==============================] - 2s 21ms/step - loss: 0.0012 - accuracy: 0.9999 - val_loss: 0.3207 - val_accuracy: 0.9805
Epoch 21/30
79/79 [==============================] - 2s 22ms/step - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.3193 - val_accuracy: 0.9794
Epoch 22/30
79/79 [==============================] - 2s 22ms/step - loss: 0.0014 - accuracy: 0.9997 - val_loss: 0.3217 - val_accuracy: 0.9794
Epoch 23/30
79/79 [==============================] - 2s 21ms/step - loss: 8.8746e-04 - accuracy: 0.9999 - val_loss: 0.3365 - val_accuracy: 0.9805
Epoch 24/30
79/79 [==============================] - 2s 21ms/step - loss: 0.0012 - accuracy: 0.9997 - val_loss: 0.3385 - val_accuracy: 0.9805
Epoch 25/30
79/79 [==============================] - 2s 21ms/step - loss: 0.0010 - accuracy: 0.9997 - val_loss: 0.3452 - val_accuracy: 0.9805
Epoch 26/30
79/79 [==============================] - 2s 21ms/step - loss: 0.0015 - accuracy: 0.9997 - val_loss: 0.3429 - val_accuracy: 0.9805
Epoch 27/30
79/79 [==============================] - 2s 21ms/step - loss: 8.6625e-04 - accuracy: 0.9999 - val_loss: 0.3336 - val_accuracy: 0.9805
Epoch 28/30
79/79 [==============================] - 2s 21ms/step - loss: 6.5785e-04 - accuracy: 0.9999 - val_loss: 0.3473 - val_accuracy: 0.9805
Epoch 29/30
79/79 [==============================] - 2s 21ms/step - loss: 5.4155e-04 - accuracy: 0.9999 - val_loss: 0.3414 - val_accuracy: 0.9794
```

```
# evaluate
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
print('Accuracy: ', score[1])
```

```
22/22 [==============================] - 0s 8ms/step - loss: 0.2782 - accuracy: 0.9786
Accuracy:  0.9785547852516174
```

CNN model

In this model we used a CNN distribution. The model took much longer to train and did not go up in accuracy very much.

```
model = models.Sequential()
model.add(layers.Embedding(10000, 128, input_length=25000))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))

model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=1e-4),  # set learning rate
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()

history = model.fit(x_train, y_train,
```

```
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
55/55 [==============================] - 1173s 21s/step - loss: 0.6974 - accuracy: 0.9548 - val_loss: 0.3276 - val_accuracy: 0.9788
Epoch 2/10
55/55 [==============================] - 1181s 21s/step - loss: 0.6974 - accuracy: 0.9548 - val_loss: 0.3276 - val_accuracy: 0.9788
Epoch 3/10
55/55 [==============================] - 1172s 21s/step - loss: 0.6974 - accuracy: 0.9548 - val_loss: 0.3276 - val_accuracy: 0.9788
Epoch 4/10
55/55 [==============================] - 1174s 21s/step - loss: 0.6974 - accuracy: 0.9548 - val_loss: 0.3276 - val_accuracy: 0.9788
Epoch 5/10
55/55 [==============================] - 1161s 21s/step - loss: 0.6974 - accuracy: 0.9548 - val_loss: 0.3276 - val_accuracy: 0.9788
Epoch 6/10
55/55 [==============================] - 1163s 21s/step - loss: 0.6974 - accuracy: 0.9548 - val_loss: 0.3276 - val_accuracy: 0.9788
Epoch 7/10
55/55 [==============================] - 1164s 21s/step - loss: 0.6974 - accuracy: 0.9548 - val_loss: 0.3276 - val_accuracy: 0.9788
Epoch 8/10
55/55 [==============================] - 1157s 21s/step - loss: 0.6974 - accuracy: 0.9548 - val_loss: 0.3276 - val_accuracy: 0.9788
Epoch 9/10
55/55 [==============================] - 1166s 21s/step - loss: 0.6974 - accuracy: 0.9548 - val_loss: 0.3276 - val_accuracy: 0.9788
Epoch 10/10
12/55 [=====>........................] - ETA: 14:53 - loss: 0.8436 - accuracy: 0.9453
```

```
# Embedding

For the final part, we added extra embedding layers but kept the model
similar to the CNN model. This one trained extremely quickly but provi
slightly lower accuracy with an average of 96%.
```

Embedding

For the final part, we added extra embedding layers but kept the model fairly similar to the CNN model. This one trained extremely quickly but provided slightly lower accuracy with an average of 96%.

```
model = models.Sequential()
model.add(layers.Embedding(10000, 8, input_length=25000))
model.add(layers.Flatten())
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 25000, 8)          80000

 flatten (Flatten)           (None, 200000)            0

 dense (Dense)               (None, 16)                3200016

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 3,280,033
Trainable params: 3,280,033
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
38/38 [==============================] - 5s 109ms/step - loss: 0.1532 - acc: 0.9511 - val_loss: 0.1954 - val_acc: 0.9636
Epoch 2/10
38/38 [==============================] - 2s 58ms/step - loss: 0.1348 - acc: 0.9760 - val_loss: 0.1739 - val_acc: 0.9636
Epoch 3/10
38/38 [==============================] - 2s 59ms/step - loss: 0.1194 - acc: 0.9760 - val_loss: 0.1498 - val_acc: 0.9636
Epoch 4/10
38/38 [==============================] - 2s 59ms/step - loss: 0.1065 - acc: 0.9760 - val_loss: 0.1444 - val_acc: 0.9636
Epoch 5/10
38/38 [==============================] - 2s 58ms/step - loss: 0.0973 - acc: 0.9760 - val_loss: 0.1505 - val_acc: 0.9636
Epoch 6/10
38/38 [==============================] - 2s 59ms/step - loss: 0.0785 - acc: 0.9760 - val_loss: 0.1361 - val_acc: 0.9636
Epoch 7/10
38/38 [==============================] - 2s 57ms/step - loss: 0.0584 - acc: 0.9768 - val_loss: 0.1419 - val_acc: 0.9636
```

```
Epoch 8/10
38/38 [==============================] - 2s 56ms/step - loss: 0.0415 - acc: 0.9809 - val_loss: 0.1827 - val_acc: 0.9636
Epoch 9/10
38/38 [==============================] - 2s 56ms/step - loss: 0.0349 - acc: 0.9851 - val_loss: 0.1760 - val_acc: 0.9636
Epoch 10/10
38/38 [==============================] - 2s 56ms/step - loss: 0.0192 - acc: 0.9950 - val_loss: 0.2014 - val_acc: 0.9636
```

Analysis

In this assignment we tried the RNN, CNN, and LSTM models. We found that RNN and LSTM needed a much larger amount of time to train than the CNN model. The sequential model and the CNN model had a similar accuracy of about 97%. However, the sequential model was much quicker. In the final test, we added additional embedding layers which increased the speed of the model but reduced the accuracy to 96.3%.