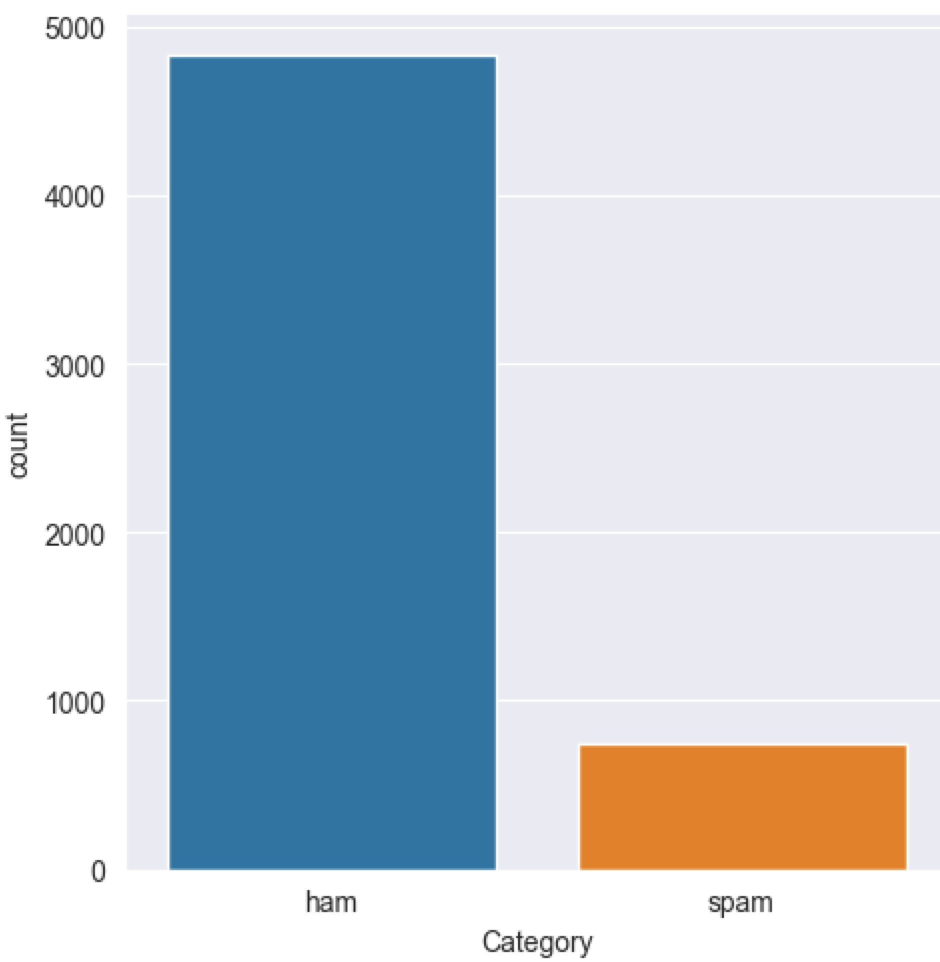
```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models, preprocessing, datasets
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
import numpy as np
import pandas as pd
import seaborn as sb
```

This dataset is similar to the ones shown in class, in contains example text messages labeled as ham or spam. Using this data, set we can train a model to be able to determine which is which.

SEQUENTIAL MODEL

```
# Print graph
df = pd.read_csv('spamtext.csv', header=0, usecols=[0,1], encoding='latin-1')
sb.catplot(x="Category" ,kind="count", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x146adef50>
```



```
# Split dataset
np.random.seed(404)
randomint = np.random.rand(len(df)) < 0.8
dftrain = df[randomint]
dftest = df[~randomint]
print("train data size: ", dftrain.shape)
print("test data size: ", dftest.shape)
```

```
train data size:  (4524, 2)
test data size:  (1048, 2)
```

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

# Fit tokenizer
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(dftrain.Message)
x_train = tokenizer.texts_to_matrix(dftrain.Message, mode='tfidf')
x_test = tokenizer.texts_to_matrix(dftest.Message, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(dftrain.Category)
y_train = encoder.transform(dftrain.Category)
y_test = encoder.transform(dftest.Category)
```

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

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

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

```
Epoch 2/30
41/41 [==============================] - 0s 9ms/step - loss: 0.2600 - accuracy: 
Epoch 3/30
41/41 [==============================] - 0s 9ms/step - loss: 0.1160 - accuracy: 
Epoch 4/30
41/41 [==============================] - 0s 9ms/step - loss: 0.0594 - accuracy: 
Epoch 5/30
41/41 [==============================] - 0s 10ms/step - loss: 0.0355 - accuracy: 
Epoch 6/30
```

```
41/41 [==============================] - 0s 9ms/step - loss: 0.0163 - accuracy:
Epoch 8/30
41/41 [==============================] - 0s 9ms/step - loss: 0.0120 - accuracy:
Epoch 9/30
41/41 [==============================] - 0s 9ms/step - loss: 0.0090 - accuracy:
Epoch 10/30
41/41 [==============================] - 0s 9ms/step - loss: 0.0069 - accuracy:
Epoch 11/30
41/41 [==============================] - 0s 9ms/step - loss: 0.0054 - accuracy:
Epoch 12/30
41/41 [==============================] - 0s 9ms/step - loss: 0.0044 - accuracy:
Epoch 13/30
41/41 [==============================] - 0s 12ms/step - loss: 0.0036 - accuracy:
Epoch 14/30
41/41 [==============================] - 0s 12ms/step - loss: 0.0031 - accuracy:
Epoch 15/30
41/41 [==============================] - 0s 11ms/step - loss: 0.0026 - accuracy:
Epoch 16/30
41/41 [==============================] - 0s 11ms/step - loss: 0.0023 - accuracy:
Epoch 17/30
41/41 [==============================] - 0s 9ms/step - loss: 0.0020 - accuracy:
Epoch 18/30
41/41 [==============================] - 0s 9ms/step - loss: 0.0018 - accuracy:
Epoch 19/30
41/41 [==============================] - 0s 10ms/step - loss: 0.0016 - accuracy:
Epoch 20/30
41/41 [==============================] - 0s 10ms/step - loss: 0.0014 - accuracy:
Epoch 21/30
41/41 [==============================] - 0s 9ms/step - loss: 0.0013 - accuracy:
Epoch 22/30
41/41 [==============================] - 0s 9ms/step - loss: 0.0012 - accuracy:
Epoch 23/30
41/41 [==============================] - 0s 9ms/step - loss: 0.0011 - accuracy:
Epoch 24/30
41/41 [==============================] - 0s 9ms/step - loss: 9.7925e-04 - accura
Epoch 25/30
41/41 [==============================] - 0s 10ms/step - loss: 9.0340e-04 - accura
Epoch 26/30
41/41 [==============================] - 0s 10ms/step - loss: 8.3376e-04 - accura
Epoch 27/30
41/41 [==============================] - 0s 11ms/step - loss: 7.7369e-04 - accura
Epoch 28/30
41/41 [==============================] - 0s 10ms/step - loss: 7.1946e-04 - accura
Epoch 29/30
41/41 [==============================] - 0s 9ms/step - loss: 6.7024e-04 - accura
Epoch 30/30
41/41 [==============================] - 0s 9ms/step - loss: 6.2651e-04 - accura
```

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

```
11/11 [==============================] - 0s 11ms/step - loss: 0.0554 - accuracy:
Accuracy:  0.9875954389572144
```

RNN Architecture

Note, I encountered a lot of trouble preprocessing my dataset, so I am unfortunately going to use the IMDB predefined one. I know this will hurt my grade, but I want to be able to turn in something as time is running short. I am very very sorry.

```
max_features = 10000
maxlen = 500
batch_size = 32

(train_data, train_labels), (test_data, test_labels) = datasets.imdb.load_data(num_wo
train_data = preprocessing.sequence.pad_sequences(train_data, maxlen=maxlen)
test_data = preprocessing.sequence.pad_sequences(test_data, maxlen=maxlen)

model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.SimpleRNN(32))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()

# Compile
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_data,
                    train_labels,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)

pred = model.predict(test_data)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(test_labels, pred))
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, None, 32)          320000

 simple_rnn_6 (SimpleRNN)    (None, 32)                2080

 dense_14 (Dense)            (None, 1)                 33

=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
```

```
Epoch 1/10
157/157 [==============================] - 13s 79ms/step - loss: 0.6572 - accura
Epoch 2/10
157/157 [==============================] - 14s 89ms/step - loss: 0.4339 - accura
Epoch 3/10
157/157 [==============================] - 13s 85ms/step - loss: 0.3252 - accura
Epoch 4/10
157/157 [==============================] - 15s 96ms/step - loss: 0.2750 - accura
Epoch 5/10
157/157 [==============================] - 13s 82ms/step - loss: 0.2027 - accura
Epoch 6/10
157/157 [==============================] - 14s 92ms/step - loss: 0.1459 - accura
Epoch 7/10
157/157 [==============================] - 14s 91ms/step - loss: 0.1016 - accura
Epoch 8/10
157/157 [==============================] - 14s 87ms/step - loss: 0.0751 - accura
Epoch 9/10
157/157 [==============================] - 14s 86ms/step - loss: 0.0452 - accura
Epoch 10/10
157/157 [==============================] - 14s 86ms/step - loss: 0.0378 - accura
782/782 [==============================] - 13s 17ms/step
              precision    recall  f1-score   support

           0       0.81      0.84      0.83     12500
           1       0.83      0.81      0.82     12500

    accuracy                           0.82     25000
   macro avg       0.82      0.82      0.82     25000
weighted avg       0.82      0.82      0.82     25000
```

LSTM

```
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.LSTM(32))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()

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

history = model.fit(train_data,
                    train_labels,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
pred = model.predict(test_data)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(test_labels, pred))
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_7 (Embedding)     (None, None, 32)          320000

 lstm (LSTM)                 (None, 32)                8320

 dense_15 (Dense)            (None, 1)                 33

=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
2023-04-22 16:25:56.387250: I tensorflow/core/common_runtime/executor.cc:1197] [
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-04-22 16:25:56.389328: I tensorflow/core/common_runtime/executor.cc:1197] [
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-04-22 16:25:56.390894: I tensorflow/core/common_runtime/executor.cc:1197] [
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
2023-04-22 16:25:56.717614: I tensorflow/core/common_runtime/executor.cc:1197] [
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-04-22 16:25:56.718935: I tensorflow/core/common_runtime/executor.cc:1197] [
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-04-22 16:25:56.720434: I tensorflow/core/common_runtime/executor.cc:1197] [
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
2023-04-22 16:25:57.329934: I tensorflow/core/common_runtime/executor.cc:1197] [
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-04-22 16:25:57.331692: I tensorflow/core/common_runtime/executor.cc:1197] [
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-04-22 16:25:57.333580: I tensorflow/core/common_runtime/executor.cc:1197] [
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
157/157 [==============================] - ETA: 0s - loss: 0.6146 - accuracy: 0.
         [[{{node gradients/split_2_grad/concat/split_2/split_dim}}]]
2023-04-22 16:26:24.097565: I tensorflow/core/common_runtime/executor.cc:1197] [
         [[{{node gradients/split_grad/concat/split/split_dim}}]]
2023-04-22 16:26:24.098625: I tensorflow/core/common_runtime/executor.cc:1197] [
         [[{{node gradients/split_1_grad/concat/split_1/split_dim}}]]
157/157 [==============================] - 30s 180ms/step - loss: 0.6146 - accura
Epoch 2/10
157/157 [==============================] - 29s 188ms/step - loss: 0.3745 - accura
Epoch 3/10
157/157 [==============================] - 28s 178ms/step - loss: 0.2821 - accura
Epoch 4/10
157/157 [==============================] - 26s 169ms/step - loss: 0.2365 - accura
Epoch 5/10
157/157 [==============================] - 30s 193ms/step - loss: 0.2016 - accura
Epoch 6/10
157/157 [==============================] - 28s 180ms/step - loss: 0.1901 - accura
Epoch 7/10
```

```
157/157 [==============================] - 30s 190ms/step - loss: 0.1649 - accur
Epoch 8/10
157/157 [==============================] - 27s 174ms/step - loss: 0.1546 - accur
Epoch 9/10
157/157 [==============================] - 29s 185ms/step - loss: 0.1398 - accur
Epoch 10/10
```

As we can see, from most to least accurate we have the dense sequential model, followed by LSTM, followed by RNN. RNN's lower accuracy is likely due to how RNN's do not perform well when working with longer series' of text. This is called the vanishing gradiant problem. LSTM's help with this problem, making them more accurate than the RNN was (even if it did take longer). Althought it was trained and tested with different data, we can clearly see that the dense sequential model is quite effective and categorizing the text.

+ Code + Text