In [2]:

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import classification_report
import numpy as np
import pandas as pd
import os
os.getcwd()
os.chdir('/Users/haochunniu/Desktop/Python/Advance AI/Final Project')
```

Problem Statement & Project Goal

In this project, I try to create different RNN models to classify various twitters in four into sentiments. The main goal of this project is to practice and emplement the important NLP models into real time cases.

Data Import

Read in the twitter dataset and fix the column names. Originally, there are 73996 twitters in the train dataset and four different seintiments (Negative, Positive, Neutral, Irrelevent). The distribution of all four sentiments is quite balance. On the other hand, there are 1000 twitters in the validation data.

In [3]:

```python
#1-1.Import train data
train=pd.read_csv('twitter_training.csv',header= None)
train.columns=['ID','Source','Label','Text']
train['ID']=list(range(len(train)))
train=train.dropna()
train_text=np.array(train['Text'])
train.head(3)
```

Out[3]:

	ID	Source	Label	Text
0	0	Borderlands	Positive	im getting on borderlands and i will murder yo...
1	1	Borderlands	Positive	I am coming to the borders and I will kill you...
2	2	Borderlands	Positive	im getting on borderlands and i will kill you ...

In [4]:

```python
round(train['Label'].value_counts()/train['Label'].value_counts().sum()*100,2)
```

Out[4]:

```
Negative      30.22
Positive      27.91
Neutral       24.47
Irrelevant    17.40
Name: Label, dtype: float64
```

In [5]:

```python
#1-2.Create dummy variables for label in train data
train_y=pd.get_dummies(train[['Label']],
                       prefix='',
                       prefix_sep='',
                       columns=['Label'])
```

In [6]:

```python
#1-3.Import test data
test=pd.read_csv('twitter_validation.csv',header=None)
test.columns=['ID','Source','Label','Text']
test['ID']=list(range(len(test)))
test_text=np.array(test['Text'])
test.head(3)
```

Out[6]:

	ID	Source	Label	Text
0	0	Facebook	Irrelevant	I mentioned on Facebook that I was struggling ...
1	1	Amazon	Neutral	BBC News - Amazon boss Jeff Bezos rejects clai...
2	2	Microsoft	Negative	@Microsoft Why do I pay for WORD when it funct...

Text Preprocessing

To train the text classification model, I need to vetorize the original text into word level input. Next, I will use general word embedding to embed the words.

In [7]:

```python
#2-1.Text pre-processing for train data
vectorize_layer = keras.layers.experimental.preprocessing.TextVectorization(
    max_tokens = None,
    standardize = 'lower_and_strip_punctuation',
    split = 'whitespace',
    ngrams = None,
    output_mode = 'int',
    output_sequence_length = None)
```

In [8]:

```python
#2-2. Apply it to the text data with "adapt". After word embedding, there are totally 42,559 unique wor
ds.
vectorize_layer.adapt(train_text)
len(vectorize_layer.get_vocabulary(10))
```

Out[8]: 42559

Classification Model1: Simple RNN

In this project, I try 2 different models, simple RNN and RNN with LSTM units. I will start from the simple RNN model.

In [9]:

```python
#3-1. Classification model of simple RNN
model_rnn = keras.Sequential()

model_rnn.add(vectorize_layer)

model_rnn.add(keras.layers.Embedding(
    input_dim = len(vectorize_layer.get_vocabulary()),
    output_dim = 256,
    mask_zero = True
))

model_rnn.add(keras.layers.SimpleRNN(256,return_sequences=True,dropout=0.3))  # see note below
model_rnn.add(keras.layers.SimpleRNN(256,dropout=0.3))
model_rnn.add(keras.layers.Dense(4, activation = 'softmax'))

#3-2. Compile the loss function for the model
model_rnn.compile(loss = keras.losses.categorical_crossentropy,
                  optimizer='adam',
                  metrics=['accuracy'])
```

In [19]:

```python
#3-3. Add early stopping layer
early_stopping = EarlyStopping(monitor='val_loss',patience=1)

#3-4. the postive and negative twitters are way more important that irrelevant and neutral
weight = {'Irrelevant': 1.,
          'Negative': 10.,
          'Neutral': 1.,
          'Positive':10.}

#3-5. Fit the model
model_rnn.fit(x = train_text, y = train_y,
              validation_split = 0.2,
              epochs=10,
              batch_size = 1024,
              callbacks=[early_stopping])
```

```
Epoch 1/10
58/58 [==============================] - 84s 1s/step - loss: 0.1139 - accuracy: 0.9559 - val_loss: 2.
7176 - val_accuracy: 0.4368
Epoch 2/10
58/58 [==============================] - 90s 2s/step - loss: 0.0971 - accuracy: 0.9615 - val_loss: 2.
8951 - val_accuracy: 0.4315
```

Out[19]:

```
<keras.callbacks.History at 0x7f8f0c0dc2d0>
```

In [25]:

```python
#3-6. Summary report of prediction on train data
train_data_result=pd.DataFrame(model_rnn.predict(train_text), columns = ['Irrelevant','Negative','Neutr
al','Positive'])
train_data_result['max']=train_data_result.max(axis = 1)
train_data_result['Label']=np.where(train_data_result['Negative']==train_data_result['max'],'Negative',
np.where(train_data_result['Positive']==train_data_result['max'],'Positive',np.where(train_data_result[
'Neutral']==train_data_result['max'],'Neutral','Irrelevant')))
print(classification_report(train['Label'], train_data_result['Label']))
```

```
              precision    recall  f1-score   support

  Irrelevant       0.81      0.87      0.84     12875
    Negative       0.91      0.85      0.88     22358
     Neutral       0.86      0.83      0.84     18108
    Positive       0.85      0.89      0.87     20655

    accuracy                           0.86     73996
   macro avg       0.86      0.86      0.86     73996
weighted avg       0.86      0.86      0.86     73996
```

In [27]:

```python
#3-7. Summary report of prediction on test data
test_data_result=pd.DataFrame(model_rnn.predict(test_text), columns = ['Irrelevant','Negative','Neutra
l','Positive'])
test_data_result['max']=test_data_result.max(axis = 1)
test_data_result['Label']=np.where(test_data_result['Negative']==test_data_result['max'],'Negative',np.
where(test_data_result['Positive']==test_data_result['max'],'Positive',np.where(test_data_result['Neutr
al']==test_data_result['max'],'Neutral','Irrelevant')))
print(classification_report(test['Label'], test_data_result['Label']))
```

```
              precision    recall  f1-score   support

  Irrelevant       0.71      0.88      0.79       172
    Negative       0.87      0.83      0.85       266
     Neutral       0.86      0.76      0.81       285
    Positive       0.86      0.87      0.87       277

    accuracy                           0.83      1000
   macro avg       0.82      0.84      0.83      1000
weighted avg       0.84      0.83      0.83      1000
```

Classification Model2: RNN with LSTM units

Next, I will try a more advance model, which is the RNN model with LSTM units.

In [28]:

```python
#4-1. Classification model of RNN with LSTM units
model_lstm = keras.Sequential()

model_lstm.add(vectorize_layer)

model_lstm.add(keras.layers.Embedding(
    input_dim = len(vectorize_layer.get_vocabulary()),
    output_dim = 512,
    mask_zero = True
))

model_lstm.add(keras.layers.LSTM(128,
                                 dropout=0.2,
                                 return_sequences=True))

model_lstm.add(keras.layers.LSTM(128,
                                 dropout=0.2,
                                 return_sequences=True))

model_lstm.add(keras.layers.LSTM(128,
                                 dropout=0.2))

model_lstm.add(keras.layers.Dense(4, activation = 'softmax'))

#4-2. Compile the loss function for the model
model_lstm.compile(loss = keras.losses.categorical_crossentropy,
                   optimizer='adam',
                   metrics=['accuracy'])
```

In [29]:

```python
#4-3. Add early stopping layer
early_stopping = EarlyStopping(monitor='val_loss',patience=1)

#4-4. the postive and negative twitters are way more important that irrelevant and neutral
weight = {'Irrelevant': 1.,
          'Negative': 2.,
          'Neutral': 1.,
          'Positive':2.}

#4-5. Fit the model
model_lstm.fit(x = train_text, y = train_y,
               validation_split = 0.2,
               epochs=10,
               batch_size = 512,
               callbacks=[early_stopping])
```

```
Epoch 1/10
116/116 [==============================] - 201s 2s/step - loss: 0.9184 - accuracy: 0.6290 - val_loss:
1.5728 - val_accuracy: 0.4520
Epoch 2/10
116/116 [==============================] - 201s 2s/step - loss: 0.3827 - accuracy: 0.8624 - val_loss:
1.9585 - val_accuracy: 0.4492
```

Out[29]:

```
<keras.callbacks.History at 0x7f8f11b21a90>
```

In [30]:

```python
#4-6. Summary report of prediction on train data
train_data_result=pd.DataFrame(model_lstm.predict(train_text), columns = ['Irrelevant','Negative','Neut
ral','Positive'])
train_data_result['max']=train_data_result.max(axis = 1)
train_data_result['Label']=np.where(train_data_result['Negative']==train_data_result['max'],'Negative',
np.where(train_data_result['Positive']==train_data_result['max'],'Positive',np.where(train_data_result[
'Neutral']==train_data_result['max'],'Neutral','Irrelevant')))
print(classification_report(train['Label'], train_data_result['Label']))
```

```
              precision    recall  f1-score   support

  Irrelevant       0.76      0.82      0.79     12875
    Negative       0.85      0.84      0.84     22358
     Neutral       0.87      0.76      0.81     18108
    Positive       0.81      0.86      0.83     20655

    accuracy                           0.82     73996
   macro avg       0.82      0.82      0.82     73996
weighted avg       0.83      0.82      0.82     73996
```

In [31]:

```python
#4-7. Summary report of prediction on test data
test_data_result=pd.DataFrame(model_lstm.predict(test_text), columns = ['Irrelevant','Negative','Neutra
l','Positive'])
test_data_result['max']=test_data_result.max(axis = 1)
test_data_result['Label']=np.where(test_data_result['Negative']==test_data_result['max'],'Negative',np.
where(test_data_result['Positive']==test_data_result['max'],'Positive',np.where(test_data_result['Neutr
al']==test_data_result['max'],'Neutral','Irrelevant')))
print(classification_report(test['Label'], test_data_result['Label']))
```

```
              precision    recall  f1-score   support

  Irrelevant       0.80      0.87      0.83       172
    Negative       0.86      0.85      0.85       266
     Neutral       0.87      0.82      0.85       285
    Positive       0.88      0.89      0.89       277

    accuracy                           0.86      1000
   macro avg       0.85      0.86      0.85      1000
weighted avg       0.86      0.86      0.86      1000
```

Classification Model3: RNN with LSTM units and BERT word embedding

Originally, I create this one layer LSTM model with BERT word embedding. Yet, I eventually notice that with my computer resource, I am not able to train the model based on this large amount of raw data. Yet, the entire process of creating this model gives me a deeper understanding of the BERT word embedding method.

In [7]:

```python
from transformers import BertTokenizer, TFBertModel

# fetch the pre-trained model (it will download a model file ~500M)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = TFBertModel.from_pretrained("bert-base-uncased")
```

```
Some layers from the model checkpoint at bert-base-uncased were not used when initializing TFBertMode
l: ['nsp___cls', 'mlm___cls']
- This IS expected if you are initializing TFBertModel from the checkpoint of a model trained on anot
her task or with another architecture (e.g. initializing a BertForSequenceClassification model from a
BertForPreTraining model).
- This IS NOT expected if you are initializing TFBertModel from the checkpoint of a model that you ex
pect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequen
ceClassification model).
All the layers of TFBertModel were initialized from the model checkpoint at bert-base-uncased.
If your task is similar to the task the model of the checkpoint was trained on, you can already use T
FBertModel for predictions without further training.
```

In [ ]:

```python
#5-1.Use BERT to encode texts
vectorized_text = tokenizer(train_text.tolist(), return_tensors='tf', padding=True)
bert_embeddings = bert_model(vectorized_text)['last_hidden_state']
```

In [ ]:

```python
#5-2.Build a single layer LSTM model with BERT embeddings
embeddings = keras.layers.Input(shape = (bert_embeddings.shape[1], bert_embeddings.shape[2]))
masked_embeddings = tf.keras.layers.Masking(mask_value=0)(embeddings)
h_all, h_final, c_final = keras.layers.LSTM(units = 128,return_state = True)(masked_embeddings)
pred = keras.layers.Dense(units = 4,activation='softmax')(h_final)
```

In [ ]:

```python
#5-3.Assemble model
model_bert_lstm = keras.Model(inputs = embeddings,outputs = pred)
```

In [ ]:

```python
# training with 20% validation and 10 epochs.
model_bert_lstm.fit(x = bert_embeddings,
                    y = train_y,
                    batch_size = 32,
                    epochs = 10,
                    validation_split = 0.2)
```