In [3]:
```python
import numpy as np
import pandas as pd

data = pd.read_csv('RNNTweets.csv')
data.head()
```

Out[3]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	ai
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	

In [4]:
```python
reviews = np.array(data['text'])[:14000]
labels = np.array(data['airline_sentiment'])[:14000]
```

In [5]:
```python
data['text'].loc[14639]
```

Out[5]:
```
'@AmericanAir we have 8 ppl so we need 2 know how many seats are on the next flight. Plz put
us on standby for 4 people on the next flight?'
```

In [6]:
```python
data['airline_sentiment'].loc[14639]
```

Out[6]:
```
'neutral'
```

In [7]:
```python
from collections import Counter

Counter(labels)
```

Out[7]:
```
Counter({'neutral': 3017, 'positive': 2304, 'negative': 8679})
```

In [8]:
```python
punctuation = '!"#$%&\'()*+,-./:;<=>?[\\]^_`{|}~'

# get rid of punctuation
all_reviews = 'separator'.join(reviews)
all_reviews = all_reviews.lower()
all_text = ''.join([c for c in all_reviews if c not in punctuation])

# split by new lines and spaces
reviews_split = all_text.split('separator')
all_text = ' '.join(reviews_split)

# create a list of words
words = all_text.split()
```

In [9]:
```python
# get rid of web address, twitter id, and digit
new_reviews = []
for review in reviews_split:
    review = review.split()
    new_text = []
    for word in review:
        if (word[0] != '@') & ('http' not in word) & (~word.isdigit()):
            new_text.append(word)
    new_reviews.append(new_text)
```

In [10]:
```python
## Build a dictionary that maps words to integers
counts = Counter(words)
vocab = sorted(counts, key=counts.get, reverse=True)
vocab_to_int = {word: ii for ii, word in enumerate(vocab, 1)}

## use the dict to tokenize each review in reviews_split
## store the tokenized reviews in reviews_ints
reviews_ints = []
for review in new_reviews:
    reviews_ints.append([vocab_to_int[word] for word in review])
```

In [11]:
```python
# stats about vocabulary
print('Unique words: ', len((vocab_to_int)))  # should ~ 74000+
print()

# print tokens in first review
print('Tokenized review: \n', reviews_ints[:1])
```

```
Unique words:  16727

Tokenized review:
 [[57, 213]]
```

In [12]:
```python
# 1=positive, 1=neutral, 0=negative label conversion
encoded_labels = []
for label in labels:
    if label == 'neutral':
        encoded_labels.append(1)
    elif label == 'negative':
        encoded_labels.append(0)
    else:
        encoded_labels.append(1)

encoded_labels = np.asarray(encoded_labels)
```

In [13]:
```python
def pad_features(reviews_ints, seq_length):
    ''' Return features of review_ints, where each review is padded with 0's
        or truncated to the input seq_length.
    '''

    # getting the correct rows x cols shape
    features = np.zeros((len(reviews_ints), seq_length), dtype=int)

    # for each review, I grab that review and
    for i, row in enumerate(reviews_ints):
        features[i, -len(row):] = np.array(row)[:seq_length]

    return features
```

In [14]:
```python
# Test implementation!

seq_length = 30

features = pad_features(reviews_ints, seq_length=seq_length)

## test statements
assert len(features)==len(reviews_ints), "The features should have as many rows as reviews."
assert len(features[0])==seq_length, "Each feature row should contain seq_length values."

# print first 10 values of the first 30 batches
print(features[:10,:10])
```

```
[[  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 446]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]]
```

In [15]:
```python
split_frac = 0.8

## split data into training, validation, and test data (features and labels, x and y)

split_idx = int(len(features)*split_frac)
train_x, remaining_x = features[:split_idx], features[split_idx:]
train_y, remaining_y = encoded_labels[:split_idx], encoded_labels[split_idx:]

test_idx = int(len(remaining_x)*0.5)
val_x, test_x = remaining_x[:test_idx], remaining_x[test_idx:]
val_y, test_y = remaining_y[:test_idx], remaining_y[test_idx:]

## print out the shapes of the resultant feature data
print("\t\t\tFeature Shapes:")
print("Train set: \t\t{}".format(train_x.shape),
      "\nValidation set: \t{}".format(val_x.shape),
      "\nTest set: \t\t{}".format(test_x.shape))
```

```
                        Feature Shapes:
Train set:              (11200, 30)
Validation set:         (1400, 30)
Test set:               (1400, 30)
```

In [16]:
```python
import torch
from torch.utils.data import TensorDataset, DataLoader

# create Tensor datasets
train_data = TensorDataset(torch.from_numpy(train_x), torch.from_numpy(train_y))
valid_data = TensorDataset(torch.from_numpy(val_x), torch.from_numpy(val_y))
test_data = TensorDataset(torch.from_numpy(test_x), torch.from_numpy(test_y))

# dataloaders
batch_size = 50

# make sure the SHUFFLE the training data
train_loader = DataLoader(train_data, shuffle=True, batch_size=batch_size)
valid_loader = DataLoader(valid_data, shuffle=True, batch_size=batch_size)
test_loader = DataLoader(test_data, shuffle=True, batch_size=batch_size)
```

In [17]:
```python
# obtain one batch of training data
dataiter = iter(train_loader)
sample_x, sample_y = dataiter.next()

print('Sample input size: ', sample_x.size()) # batch_size, seq_length
print('Sample input: \n', sample_x)
print()
print('Sample label size: ', sample_y.size()) # batch_size
print('Sample label: \n', sample_y)
```

```
Sample input size:  torch.Size([50, 30])
Sample input:
 tensor([[    0,     0,     0,  ...,     4,    50,   771],
        [    0,     0,     0,  ...,    93,   194,    99],
        [    0,     0,     0,  ...,     9,    47, 14866],
        ...,
        [    0,     0,     0,  ...,   117,    32,  1992],
        [    0,     0,     0,  ...,    35,     4,   523],
        [    0,     0,     0,  ...,    34,     5,    46]], dtype=torch.int32)

Sample label size:  torch.Size([50])
Sample label:
 tensor([0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1,
        0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 1], dtype=torch.int32)
```

In [18]:
```python
# First checking if GPU is available
train_on_gpu=torch.cuda.is_available()

if(train_on_gpu):
    print('Training on GPU.')
else:
    print('No GPU available, training on CPU.')
```

```
No GPU available, training on CPU.
```

In [19]:
```python
import torch.nn as nn

class SentimentRNN(nn.Module):
    """
    The RNN model that will be used to perform Sentiment analysis.
    """

    def __init__(self, vocab_size, output_size, embedding_dim, hidden_dim, n_layers, drop_pr
ob=0.5):
        """
        Initialize the model by setting up the layers.
        """
        super(SentimentRNN, self).__init__()

        self.output_size = output_size
        self.n_layers = n_layers
        self.hidden_dim = hidden_dim

        # embedding and LSTM layers
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, n_layers,
                            dropout=drop_prob, batch_first=True)

        # dropout layer
        self.dropout = nn.Dropout(0.3)

        # linear and sigmoid layers
        self.fc = nn.Linear(hidden_dim, output_size)
        self.sig = nn.Sigmoid()


    def forward(self, x, hidden):
        """
        Perform a forward pass of our model on some input and hidden state.
        """
        batch_size = x.size(0)

        # embeddings and lstm_out
        x = x.long()
        embeds = self.embedding(x)
        lstm_out, hidden = self.lstm(embeds, hidden)

        # stack up lstm outputs
        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

        # dropout and fully-connected layer
        out = self.dropout(lstm_out)
        out = self.fc(out)
        # sigmoid function
        sig_out = self.sig(out)

        # reshape to be batch_size first
        sig_out = sig_out.view(batch_size, -1)
        sig_out = sig_out[:, -1] # get last batch of labels

        # return last sigmoid output and hidden state
        return sig_out, hidden


    def init_hidden(self, batch_size):
        ''' Initializes hidden state '''
        # Create two new tensors with sizes n_layers x batch_size x hidden_dim,
        # initialized to zero, for hidden state and cell state of LSTM
        weight = next(self.parameters()).data

        if (train_on_gpu):
            hidden = (weight.new(self.n_layers, batch_size, self.hidden_dim).zero_().cuda(),
                  weight.new(self.n_layers, batch_size, self.hidden_dim).zero_().cuda())
        else:
            hidden = (weight.new(self.n_layers, batch_size, self.hidden_dim).zero_(),
                      weight.new(self.n_layers, batch_size, self.hidden_dim).zero_())

        return hidden
```

In [20]:
```python
# Instantiate the model w/ hyperparams
vocab_size = len(vocab_to_int)+1 # +1 for the 0 padding + our word tokens
output_size = 1
embedding_dim = 200
hidden_dim = 128
n_layers = 2

net = SentimentRNN(vocab_size, output_size, embedding_dim, hidden_dim, n_layers)

print(net)
```

```
SentimentRNN(
  (embedding): Embedding(16728, 200)
  (lstm): LSTM(200, 128, num_layers=2, batch_first=True, dropout=0.5)
  (dropout): Dropout(p=0.3)
  (fc): Linear(in_features=128, out_features=1, bias=True)
  (sig): Sigmoid()
)
```

In [21]:
```python
# loss and optimization functions
lr=0.001

criterion = nn.BCELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=lr)
```

In [22]:
```python
# training params

epochs = 10

counter = 0
print_every = 100
clip=5 # gradient clipping

# move model to GPU, if available
if(train_on_gpu):
    net.cuda()

net.train()
# train for some number of epochs
for e in range(epochs):
    # initialize hidden state
    h = net.init_hidden(batch_size)

    # batch loop
    for inputs, labels in train_loader:
        counter += 1

        if(train_on_gpu):
            inputs, labels = inputs.cuda(), labels.cuda()

        # Creating new variables for the hidden state, otherwise
        # we'd backprop through the entire training history
        h = tuple([each.data for each in h])

        # zero accumulated gradients
        net.zero_grad()

        # get the output from the model
        output, h = net(inputs, h)

        # calculate the loss and perform backprop
        loss = criterion(output.squeeze(), labels.float())
        loss.backward()
        # `clip_grad_norm` helps prevent the exploding gradient problem in RNNs / LSTMs.
        nn.utils.clip_grad_norm_(net.parameters(), clip)
        optimizer.step()

        # loss stats
        if counter % print_every == 0:
            # Get validation loss
            val_h = net.init_hidden(batch_size)
            val_losses = []
            net.eval()
            for inputs, labels in valid_loader:

                # Creating new variables for the hidden state, otherwise
                # we'd backprop through the entire training history
                val_h = tuple([each.data for each in val_h])

                if(train_on_gpu):
                    inputs, labels = inputs.cuda(), labels.cuda()

                output, val_h = net(inputs, val_h)
                val_loss = criterion(output.squeeze(), labels.float())

                val_losses.append(val_loss.item())

            net.train()
            print("Epoch: {}/{}...".format(e+1, epochs),
                  "Step: {}...".format(counter),
                  "Loss: {:.6f}...".format(loss.item()),
                  "Val Loss: {:.6f}".format(np.mean(val_losses)))
```

```
Epoch: 1/10... Step: 100... Loss: 0.551524... Val Loss: 0.480843
Epoch: 1/10... Step: 200... Loss: 0.362361... Val Loss: 0.474334
Epoch: 2/10... Step: 300... Loss: 0.298605... Val Loss: 0.439574
Epoch: 2/10... Step: 400... Loss: 0.386869... Val Loss: 0.440207
Epoch: 3/10... Step: 500... Loss: 0.201143... Val Loss: 0.460960
Epoch: 3/10... Step: 600... Loss: 0.164055... Val Loss: 0.435676
Epoch: 4/10... Step: 700... Loss: 0.076796... Val Loss: 0.573624
Epoch: 4/10... Step: 800... Loss: 0.202839... Val Loss: 0.573702
Epoch: 5/10... Step: 900... Loss: 0.106682... Val Loss: 0.525078
Epoch: 5/10... Step: 1000... Loss: 0.086866... Val Loss: 0.695109
Epoch: 5/10... Step: 1100... Loss: 0.316709... Val Loss: 0.609379
Epoch: 6/10... Step: 1200... Loss: 0.184955... Val Loss: 0.760722
Epoch: 6/10... Step: 1300... Loss: 0.088183... Val Loss: 0.778086
Epoch: 7/10... Step: 1400... Loss: 0.011401... Val Loss: 0.898175
Epoch: 7/10... Step: 1500... Loss: 0.012297... Val Loss: 0.902463
Epoch: 8/10... Step: 1600... Loss: 0.066672... Val Loss: 0.868812
Epoch: 8/10... Step: 1700... Loss: 0.022037... Val Loss: 0.974010
Epoch: 9/10... Step: 1800... Loss: 0.035473... Val Loss: 0.871920
Epoch: 9/10... Step: 1900... Loss: 0.006784... Val Loss: 1.015922
Epoch: 9/10... Step: 2000... Loss: 0.003342... Val Loss: 1.019821
Epoch: 10/10... Step: 2100... Loss: 0.013238... Val Loss: 1.015664
Epoch: 10/10... Step: 2200... Loss: 0.018347... Val Loss: 1.002716
```

In [24]:
```python
# Get test data loss and accuracy

test_losses = [] # track loss
num_correct = 0

# init hidden state
h = net.init_hidden(batch_size)

net.eval()
# iterate over test data
for inputs, labels in test_loader:

    # Creating new variables for the hidden state, otherwise
    # we'd backprop through the entire training history
    h = tuple([each.data for each in h])

    if(train_on_gpu):
        inputs, labels = inputs.cuda(), labels.cuda()

    # get predicted outputs
    output, h = net(inputs, h)

    # calculate loss
    test_loss = criterion(output.squeeze(), labels.float())
    test_losses.append(test_loss.item())

    # convert output probabilities to predicted class (0 or 1)
    pred = torch.round(output.squeeze())  # rounds to the nearest integer

    # compare predictions to true label
    correct_tensor = pred.eq(labels.float().view_as(pred))
    correct = np.squeeze(correct_tensor.numpy()) if not train_on_gpu else np.squeeze(correct
_tensor.cpu().numpy())
    num_correct += np.sum(correct)


# -- stats! -- ##
# avg test loss
print("Test loss: {:.3f}".format(np.mean(test_losses)))

# accuracy over all test data
test_acc = num_correct/len(test_loader.dataset)
print("Test accuracy: {:.3f}".format(test_acc))
```

```
Test loss: 0.746
Test accuracy: 0.832
```

In [25]:
```python
# negative test review
test_review = "@AmericanAir you have my money, you change my flight, and don't answer your p
hones! Any other suggestions so I can make my commitment??"
```

In [26]:
```python
def tokenize_review(test_review):
    test_review = test_review.lower() # lowercase
    # get rid of punctuation
    test_text = ''.join([c for c in test_review if c not in punctuation])

    # splitting by spaces
    test_words = test_text.split()

    # get rid of web address, twitter id, and digit
    new_text = []
    for word in test_words:
        if (word[0] != '@') & ('http' not in word) & (~word.isdigit()):
            new_text.append(word)

    # tokens
    test_ints = []
    test_ints.append([vocab_to_int[word] for word in new_text])

    return test_ints

# test code and generate tokenized review
test_ints = tokenize_review(test_review)
print(test_ints)
```

```
[[5, 22, 11, 367, 5, 126, 11, 8, 10, 85, 335, 21, 922, 93, 194, 1550, 44, 3, 34, 125, 11, 288
8]]
```

In [27]:
```python
# test sequence padding
seq_length=30
features = pad_features(test_ints, seq_length)

print(features)
```

```
[[   0    0    0    0    0    0    0    0    5   22   11  367    5  126
    11    8   10   85  335   21  922   93  194 1550   44    3   34  125
    11 2888]]
```

In [28]:
```python
# test conversion to tensor and pass into your model
feature_tensor = torch.from_numpy(features)
print(feature_tensor.size())
```

```
torch.Size([1, 30])
```

In [29]:
```python
def predict(net, test_review, sequence_length=30):

    net.eval()

    # tokenize review
    test_ints = tokenize_review(test_review)

    # pad tokenized sequence
    seq_length=sequence_length
    features = pad_features(test_ints, seq_length)

    # convert to tensor to pass into your model
    feature_tensor = torch.from_numpy(features)

    batch_size = feature_tensor.size(0)

    # initialize hidden state
    h = net.init_hidden(batch_size)

    if(train_on_gpu):
        feature_tensor = feature_tensor.cuda()

    # get the output from the model
    output, h = net(feature_tensor, h)

    # convert output probabilities to predicted class (0 or 1)
    pred = torch.round(output.squeeze())
    # printing output value, before rounding
    print('Prediction value, pre-rounding: {:.6f}'.format(output.item()))

    # print custom response
    if(pred.item()==1):
        print("Non-negative review detected.")
    else:
        print("Negative review detected.")
```

In [30]:
```python
seq_length = 30 # good to use the length that was trained on
```

In [31]:
```python
# call function on negative review
test_review_neg = "@AmericanAir you have my money, you change my flight, and don't answer yo
ur phones! Any other suggestions so I can make my commitment??"
predict(net, test_review_neg, seq_length)
```

```
Prediction value, pre-rounding: 0.001016
Negative review detected.
```

In [32]:
```python
# call function on positive review
test_review_pos = "@AmericanAir thank you we got on a different flight to Chicago."
predict(net, test_review_pos, seq_length)
```

```
Prediction value, pre-rounding: 0.999730
Non-negative review detected.
```

In [33]:
```python
# call function on neutral review
test_review_neu = "@AmericanAir i need someone to help me out"
predict(net, test_review_neu, seq_length)
```

```
Prediction value, pre-rounding: 0.926614
Non-negative review detected.
```

In [ ]: