

Predicting movie sentiments

Here we will create a model using LSTM RNN network to predict the sentiments of movie reviews present in IMDB. The movie reviews are either positive or negative.

Importing necessary package

```
In [3]: import numpy as np
import torch
from torch import nn, optim
from string import punctuation
from collections import Counter
```

Importing directory and files for our model

```
In [4]: dirc1 = "D:/1_FINAL_UNIVERSITY_TAMU/Deep Learning Udacity/deep-learning-v2-pytorch-master"
dirc2 = "/deep-learning-v2-pytorch-master/sentiment-rnn/data/"
dirc = dirc1 + dirc2
```

```
In [5]: with open(dirc+"reviews.txt", "r") as f:
reviews = f.read()
with open(dirc+"labels.txt", "r") as f:
labels = f.read()
```

Printing some contents of our files

```
In [6]: print(reviews[:1000])
print(labels[:10])
```

bromwell high is a cartoon comedy . it ran at the same time as some other programs about school life such as teachers . my years in the teaching profession lead me to believe that bromwell high's satire is much closer to reality than is teachers . the scramble to survive financially the insightful students who can see right through their pathetic teachers' pomp the pettiness of the whole situation all remind me of the schools i knew and their students . when i saw the episode in which a student repeatedly tried to burn down the school i immediately recalled at high . a classic line inspector i'm here to sack one of your teachers . student welcome to bromwell high . i expect that many adults of my age think that bromwell high is far fetched . what a pity that it isn't

story of a man who has unnatural feelings for a pig . starts out with a opening scene that is a terrific example of absurd comedy . a formal orchestra audience is turned positive
n

Formating

in the next few steps we will format our text file so taht we can train our model effectively.

The formatting steps will include:

1. Getting rid of punctuation and then splitting the text into words
2. Creating a dictionary of words that will represent our vocabulary
3. Assigning a number to the vocabulary in order to send it to our model
4. Assigning every review in our review list to a number based on our vocabulary
5. Padding and cutting the overused words and getting rid of zero reviews, for making inputs of same size

```
In [7]: reviews = reviews.lower()
all_text = ''.join([c for c in reviews if c not in punctuation])
all_text_split = all_text.split("\n")
all_text_mod = ' '.join(all_text_split)
words = all_text_mod.split()
print(words[:100])
```

```
['bromwell', 'high', 'is', 'a', 'cartoon', 'comedy', 'it', 'ran', 'at', 'the', 'same', 'time', 'as', 'some', 'other', 'pr  
ograms', 'about', 'school', 'life', 'such', 'as', 'teachers', 'my', 'years', 'in', 'the', 'teaching', 'profession', 'lea  
d', 'me', 'to', 'believe', 'that', 'bromwell', 'high', 's', 'satire', 'is', 'much', 'closer', 'to', 'reality', 'than', 'i  
s', 'teachers', 'the', 'scramble', 'to', 'survive', 'financially', 'the', 'insightful', 'students', 'who', 'can', 'see',  
'right', 'through', 'their', 'pathetic', 'teachers', 'pomp', 'the', 'pettiness', 'of', 'the', 'whole', 'situation', 'al  
l', 'remind', 'me', 'of', 'the', 'schools', 'i', 'knew', 'and', 'their', 'students', 'when', 'i', 'saw', 'the', 'episod  
e', 'in', 'which', 'a', 'student', 'repeatedly', 'tried', 'to', 'burn', 'down', 'the', 'school', 'i', 'immediately', 'rec  
alled', 'at', 'high']
```

```
In [8]: word_count = Counter(words).most_common()
print(word_count[:10])
vocab2int = dict()
for i in range(len(word_count)):
    k = word_count[i][0]
    vocab2int[k] = i+1
```

```

reviews_int = []
for review in all_text_split:
    reviews_int.append([vocab2int[word] for word in review.split()])

print(reviews_int[:1])

```

```

[('the', 336713), ('and', 164107), ('a', 163009), ('of', 145864), ('to', 135720), ('is', 107328), ('br', 101872), ('it',
96352), ('in', 93968), ('i', 87623)]
[[21025, 308, 6, 3, 1050, 207, 8, 2138, 32, 1, 171, 57, 15, 49, 81, 5785, 44, 382, 110, 140, 15, 5194, 60, 154, 9, 1, 497
5, 5852, 475, 71, 5, 260, 12, 21025, 308, 13, 1978, 6, 74, 2395, 5, 613, 73, 6, 5194, 1, 24103, 5, 1983, 10166, 1, 5786,
1499, 36, 51, 66, 204, 145, 67, 1199, 5194, 19869, 1, 37442, 4, 1, 221, 883, 31, 2988, 71, 4, 1, 5787, 10, 686, 2, 67, 14
99, 54, 10, 216, 1, 383, 9, 62, 3, 1406, 3686, 783, 5, 3483, 180, 1, 382, 10, 1212, 13583, 32, 308, 3, 349, 341, 2913, 1
0, 143, 127, 5, 7690, 30, 4, 129, 5194, 1406, 2326, 5, 21025, 308, 10, 528, 12, 109, 1448, 4, 60, 543, 102, 12, 21025, 30
8, 6, 227, 4146, 48, 3, 2211, 12, 8, 215, 23]]

```

```

In [9]: print('Unique words: ', len((vocab2int))) # should ~ 74000+
print()

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

```

Unique words: 74072

Tokenized review:

```

[[21025, 308, 6, 3, 1050, 207, 8, 2138, 32, 1, 171, 57, 15, 49, 81, 5785, 44, 382, 110, 140, 15, 5194, 60, 154, 9, 1, 49
75, 5852, 475, 71, 5, 260, 12, 21025, 308, 13, 1978, 6, 74, 2395, 5, 613, 73, 6, 5194, 1, 24103, 5, 1983, 10166, 1, 5786,
1499, 36, 51, 66, 204, 145, 67, 1199, 5194, 19869, 1, 37442, 4, 1, 221, 883, 31, 2988, 71, 4, 1, 5787, 10, 686, 2, 67, 14
99, 54, 10, 216, 1, 383, 9, 62, 3, 1406, 3686, 783, 5, 3483, 180, 1, 382, 10, 1212, 13583, 32, 308, 3, 349, 341, 2913, 1
0, 143, 127, 5, 7690, 30, 4, 129, 5194, 1406, 2326, 5, 21025, 308, 10, 528, 12, 109, 1448, 4, 60, 543, 102, 12, 21025, 30
8, 6, 227, 4146, 48, 3, 2211, 12, 8, 215, 23]]

```

```

In [10]: label_list = np.array(labels.split("\n"))
# print(label_list[:100])
encoded_labels = np.where(label_list=="positive",1,0)
# print(encoded_labels[:100])

```

```

In [11]: review_lens = Counter([len(x) for x in reviews_int])
print("Zero-length reviews: {}".format(review_lens[0]))
print("Maximum review length: {}".format(max(review_lens)))

```

Zero-length reviews: 1

Maximum review length: 2514

```
In [20]: print("Number of reviews before removing outlier:", len(reviews_int))

non_zero_idx = [i for i, review in enumerate(reviews_int) if len(review) != 0]

reviews_int = [reviews_int[i] for i in non_zero_idx]
encoded_labels = np.array([encoded_labels[i] for i in non_zero_idx])

print("Number of reviews after removing outlier:", len(reviews_int))
```

Number of reviews before removing outlier: 25000
Number of reviews after removing outlier: 25000

```
In [13]: def pad_features(reviews_int, seq_length):

    features = np.zeros((len(reviews_int), seq_length), dtype=int)

    for i, row in enumerate(reviews_int):
        features[i, -len(row):] = np.array(row)[:seq_length]

    return features
```

```
In [14]: seq_length = 200

features = pad_features(reviews_int, seq_length=seq_length)

## test statements - do not change - ##
assert len(features) == len(reviews_int), "Your 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[:30, :10])
```

```
[[ 0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0]
 [22382  42 46418  15  706 17139 3389  47  77  35]
 [ 4505  505  15  3 3342  162 8312 1652  6 4819]
 [ 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]
[ 54 10 14 116 60 798 552 71 364 5]
[ 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]
[ 1 330 578 34 3 162 748 2731 9 325]
[ 9 11 10171 5305 1946 689 444 22 280 673]
[ 0 0 0 0 0 0 0 0 0 0]
[ 1 307 10399 2069 1565 6202 6528 3288 17946 10628]
[ 0 0 0 0 0 0 0 0 0 0]
[ 21 122 2069 1565 515 8181 88 6 1325 1182]
[ 1 20 6 76 40 6 58 81 95 5]
[ 54 10 84 329 26230 46427 63 10 14 614]
[ 11 20 6 30 1436 32317 3769 690 15100 6]
[ 0 0 0 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 0 0 0 0]
[ 40 26 109 17952 1422 9 1 327 4 125]
[ 0 0 0 0 0 0 0 0 0 0]
[ 10 499 1 307 10399 55 74 8 13 30]
[ 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]]
```

Deviding the dataset into Train, Validation and Test Dataset

In [50]:

```
# split_frac = 0.8

# train_idx = int(len(reviews_int)*split_frac)
# train_x = pad_features(reviews_int[:train_idx],seq_length)
# val_test_data = reviews_int[train_idx:]
# val_x = pad_features(val_test_data[:len(val_test_data)//2],seq_length)
# test_x = pad_features(val_test_data[len(val_test_data)//2:],seq_length)

# train_y = encoded_labels[:train_idx]
# val_test_labels = encoded_labels[train_idx:]
# val_y = val_test_labels[:len(val_test_labels)//2]
# test_y = val_test_labels[len(val_test_labels)//2:]

# print(train_x.shape)
# print(val_x.shape)
# print(test_x.shape)

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 your 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:	(20000, 200)
Validation set:	(2500, 200)
Test set:	(2500, 200)

```

In [51]: print(train_y.shape)
         print(val_y.shape)
         print(test_y.shape)

```

```

(20000,)
(2500,)
(2500,)

```

```

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

```

```

In [52]: 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))

         batch_size = 50

         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 [53]: # 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, 200])
Sample input:
tensor([[ 0,  0,  0, ..., 35014,  520,  18],
        [ 0,  0,  0, ...,  38,  645, 1722],
        [1175, 45, 40, ...,  6,  44,  1],
        ...,
        [ 0,  0,  0, ...,  13,  24,  76],
        [ 0,  0,  0, ..., 571,  5,  21],
        [ 11, 18, 285, ..., 3278, 45, 25]], dtype=torch.int32)
```

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

```
In [54]: train_on_gpu=torch.cuda.is_available()

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

Training on GPU.

Defining Our Model

We are going to use our padded features and then pass it through our LSTM network with `n_layers` and `hidden_dim` and then through fully connected layer to predict the sentiment.

```
In [55]: class SentimentRNN(nn.Module):
```

```

def __init__(self, vocab_size, hidden_dim, n_layers, embedding_dim, output_size, drop_prob = 0.5):

    super(SentimentRNN, self).__init__()

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

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

    self.dropout = nn.Dropout(0.3)

    self.fc = nn.Linear(hidden_dim, output_size)
    self.sig = nn.Sigmoid()

def forward(self, x, hidden):

    batch_size = x.size(0)

    x = x.long()

    embeds = self.embedding(x)
    out, hidden = self.lstm(embeds, hidden)
    out = out[:, -1, :]

    out = self.dropout(out)
    out = self.fc(out)
    sig_out = self.sig(out)

    return sig_out, hidden

def init_hidden(self, batch_size):

    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
```

Initialising parameters

```
In [56]: # Instantiate the model w/ hyperparams
vocab_size = len(vocab2int)+1 # +1 for the 0 padding + our word tokens
output_size = 1
embedding_dim = 400
hidden_dim = 256
n_layers = 2

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

print(net)

SentimentRNN(
  (embedding): Embedding(74073, 400)
  (lstm): LSTM(400, 256, num_layers=2, batch_first=True, dropout=0.5)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=256, out_features=1, bias=True)
  (sig): Sigmoid()
)
```

Defining loss and optimisation step

```
In [57]: lr=0.001

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

Training our model

We will train our model and then validate on validation data

```
In [58]: epochs = 5
```

```

counter = 0
print_every = 100
clip = 5

if train_on_gpu:
    net.cuda()

net.train()
for e in range(epochs):
    h = net.init_hidden(batch_size)
    for inputs, labels in train_loader:
        counter += 1
        if train_on_gpu:
            inputs, labels = inputs.cuda(), labels.cuda()

        h = [each.data for each in h]

        net.zero_grad()

        output, h = net(inputs, h)

        loss = criterion(output.squeeze(), labels.float())

        loss.backward()

        nn.utils.clip_grad_norm_(net.parameters(), clip)
        optimizer.step()

    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: 1/5... Step: 100... Loss: 0.649008... Val Loss: 0.642537
Epoch: 1/5... Step: 200... Loss: 0.741309... Val Loss: 0.662694
Epoch: 1/5... Step: 300... Loss: 0.495049... Val Loss: 0.654064
Epoch: 1/5... Step: 400... Loss: 0.461039... Val Loss: 0.547594
Epoch: 2/5... Step: 500... Loss: 0.421245... Val Loss: 0.604322
Epoch: 2/5... Step: 600... Loss: 0.424554... Val Loss: 0.483509
Epoch: 2/5... Step: 700... Loss: 0.356222... Val Loss: 0.457068
Epoch: 2/5... Step: 800... Loss: 0.381022... Val Loss: 0.449615
Epoch: 3/5... Step: 900... Loss: 0.337756... Val Loss: 0.450060
Epoch: 3/5... Step: 1000... Loss: 0.229221... Val Loss: 0.446776
Epoch: 3/5... Step: 1100... Loss: 0.185805... Val Loss: 0.448915
Epoch: 3/5... Step: 1200... Loss: 0.205090... Val Loss: 0.478299
Epoch: 4/5... Step: 1300... Loss: 0.201544... Val Loss: 0.516306
Epoch: 4/5... Step: 1400... Loss: 0.147721... Val Loss: 0.588707
Epoch: 4/5... Step: 1500... Loss: 0.373541... Val Loss: 0.532522
Epoch: 4/5... Step: 1600... Loss: 0.175939... Val Loss: 0.529818
Epoch: 5/5... Step: 1700... Loss: 0.092958... Val Loss: 0.626706
Epoch: 5/5... Step: 1800... Loss: 0.075789... Val Loss: 0.644308
Epoch: 5/5... Step: 1900... Loss: 0.193247... Val Loss: 0.808672
Epoch: 5/5... Step: 2000... Loss: 0.021674... Val Loss: 0.657571

```

Testing our model

In [59]:

```

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: 0.679
Test accuracy: 0.792

Testing on an unseen text

In [60]: `test_review_neg = 'The worst movie I have seen; acting was terrible and I want my money back. This movie had bad acting a`

Proceeding same as before

Formating, then passing through the model

```
In [61]: from string import punctuation
```

```
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()

    # tokens
    test_ints = []
    test_ints.append([vocab2int.get(word, 0) for word in test_words])

    return test_ints

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

```
[[1, 247, 18, 10, 28, 108, 113, 14, 388, 2, 10, 181, 60, 273, 144, 11, 18, 68, 76, 113, 2, 1, 410, 14, 539]]
```

```
In [62]:
```

```
seq_length=200
features = pad_features(test_ints, seq_length)

print(features)
```

```
[[ 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  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  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  1 247 18 10 28
108 113 14 388  2 10 181 60 273 144 11 18 68 76 113  2  1 410
14 539]]
```

```
In [63]:
```

```
feature_tensor = torch.from_numpy(features)
print(feature_tensor.size())
```

```
torch.Size([1, 200])
```

Creating a function to predict the output

```
In [64]: def predict(net, test_review, sequence_length=200):

    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("Positive review detected!")
    else:
        print("Negative review detected.")
```

```
In [65]: # positive test review
test_review_pos = 'This movie had the best acting and the dialogue was so good. I loved it.'
```

Predicted result

```
In [66]: # call function  
seq_length=200 # good to use the length that was trained on  
  
predict(net, test_review_neg, seq_length)
```

Prediction value, pre-rounding: 0.001236
Negative review detected.

In []: