CSCI544 - Homework Assignment 2

In [1]:
```
import torch
import torchvision
import pandas as pd
import numpy as np
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
```

Reading Data

In [2]:
```
#Reading Data from file and extratcing review_body and star_ratings only
df = pd.read_csv('data.tsv', sep='\t', error_bad_lines=False, warn_bad_lines=False, lo
data = df[['review_body', 'star_rating']]
data = data[data['review_body'].str.len() > 0 ]
```

Dataset Generation and Data Cleaning

In [3]:
```
import re
import contractions

data['star_rating'] = data['star_rating'].apply(int)

#Separating the data based on star_rating

rating_1 = data[data['star_rating']==1]
rating_2 = data[data['star_rating']==2]
rating_3 = data[data['star_rating']==3]
rating_4 = data[data['star_rating']==4]
rating_5 = data[data['star_rating']==5]



#Taking a sample of 20000 from each rating value

rating_1 = rating_1.sample(n=20000)
rating_2 = rating_2.sample(n=20000)
rating_3 = rating_3.sample(n=20000)
rating_4 = rating_4.sample(n=20000)
rating_5 = rating_5.sample(n=20000)

#Forming a dataset of 100,000 rows of balanced ratings
review_data = pd.concat([rating_1, rating_2,rating_3, rating_4,rating_5])

#Function for data cleaning
def clean_review(text):
    regex_alpha = '[^a-zA-Z]'
    regex_html = '<.*?>'
    regex_url = r'http[s]?://\S+'
    #Remove URL from reviews
    text = re.sub(r"http\S+", "", text)
    #Remove HTML from reviews
    text = re.sub(regex_html, ' ', text)
    #Convert to lower letters
    text = text.lower()
    #Fix Contractions
    text = contractions.fix(text)
    #Remove non alphabetical characters
    text = re.sub(regex_alpha, ' ', text)
    text = " ".join(text.split())
    return text


review_data["review_body"] = review_data["review_body"].apply(clean_review)
review_data = review_data.sample(frac = 1)
review_data["review_body"] = review_data["review_body"].apply(str)

X = review_data.review_body
Y = review_data.star_rating

#Divide data into train and test set, train set has 80% of data and test set has 20% d
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.2)
```

Word Embedding - Google New Word2Vec

Loaded Google Word2Vec model and performed semantic similarities for 3 examples as shown below

In [4]:
```
#Downloading google news word2vec model
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
```

In [5]:
```
from numpy import dot
from numpy.linalg import norm

#Vector for king
vec_king = wv['king']
vec_queen = wv['queen']
vec_man = wv['man']
vec_woman = wv['woman']

queen = vec_king-vec_man+vec_woman

cos_sim1 = dot(vec_queen, queen)/(norm(vec_queen)*norm(queen))
print(cos_sim1)

vec_beautiful = wv['attractive']
vec_attractive = wv['beautiful']

cos_sim2 = dot(vec_beautiful, vec_attractive)/(norm(vec_beautiful)*norm(vec_attractive
print(cos_sim2)

vec_good = wv['sweet']
vec_bad = wv['player']

cos_sim3 = dot(vec_good, vec_bad)/(norm(vec_good)*norm(vec_bad))
print(cos_sim3)
```

```
0.7300517
0.4135858
0.08265579
```

Word Embedding - Word2Vec Model using own dataset

In [6]:
```
from gensim import utils
import gensim.models

#Building own word2vec model with review data. Reference: https://radimrehurek.com/ge
class MyCorpus:
    def __iter__(self):
        for line in X:
            yield utils.simple_preprocess(line)

sentences = MyCorpus()
model = gensim.models.Word2Vec(sentences=sentences, vector_size=300, window=11,  min_c
```

In [7]:
```
vec_king1 = model.wv['king']
vec_queen1 = model.wv['queen']
vec_man1 = model.wv['man']
vec_woman1 = model.wv['woman']

queen1 = vec_king1-vec_man1+vec_woman1

cos_sim1 = dot(vec_queen1, queen1)/(norm(vec_queen1)*norm(queen1))
print(cos_sim1)

vec_beautiful1 = model.wv['beautiful']
vec_attractive1 = model.wv['attractive']

cos_sim2 = dot(vec_beautiful1, vec_attractive1)/(norm(vec_beautiful1)*norm(vec_attract
print(cos_sim2)

vec_good1 = model.wv['sweet']
vec_bad1 = model.wv['player']

cos_sim3 = dot(vec_good1, vec_bad1)/(norm(vec_good1)*norm(vec_bad1))
print(cos_sim3)
```

```
0.36439443
0.5834363
0.41000557
```

Conclusion/Answers to Q2

I see vectors generated by the pre-trained model maintained the semantic similarities better becuase our word2vec model was trained using less data. More the data, better is the model. From the cosine similarites for operations King-Man+Woman and Queen, the similarity is more with Google Word2Vec model. Both models maintained similar semantic similarity for words beautiful and attractive. But for words sweet and player which is not related and hardly seen together in context, pretrained model perfomed better to give the low value since they are not related but my model gave a higher similarity value for that. So Google Word2Vec model performed better for my example and that's because it is trained on more data.

Q3 - Feature Extraction using Word2Vec and Simple Models

For this question, I extracted features for reviews using average Word2Vec vectors for each review as the input. The feature set generated using average Word2Vec is used for training the Perceptron and Support Vector machine model and predictions are made. Below I'm printing the accuracies and comparing the accuracies with same model with features from TF-IDF from previous assignment and Word2Vec features we extracted here

In [8]:
```
#Form feature vectors for training SVM and Perceptron

def extractFeature(review):
    words = review.split(" ")
    sentence_vector = np.zeros(300)
    for word in words:
        if word in  wv.key_to_index:
            word_vector = wv[word]
            sentence_vector = sentence_vector + word_vector
    sentence_vector = sentence_vector / len(words)
    return sentence_vector

X_train_feature = list(X_train.apply(extractFeature))
#print(len(X_train_feature))
X_test_feature = list(X_test.apply(extractFeature))
#print(len(X_test_feature))

X_train_feature1 = X_train.apply(extractFeature)
X_test_feature1 = X_test.apply(extractFeature)
```

Perceptron

In [9]:
```
from sklearn.linear_model import Perceptron
from sklearn import metrics

perceptron = Perceptron(
                max_iter=250,
                tol=0.001)
perceptron.fit(list(X_train_feature), Y_train)
prediction_perceptron = perceptron.predict(list(X_test_feature))
perceptron_results = metrics.classification_report(Y_test, prediction_perceptron, outp
accuracy = str(round(perceptron_results['accuracy'],2))

print("Accuracy of Perceptron Model using TF-IDF feature: 0.39 ")
print("Accuracy of Perceptron Model using Word2Vec feature:", accuracy)
```

```
Accuracy of Perceptron Model using TF-IDF feature: 0.39
Accuracy of Perceptron Model using Word2Vec feature: 0.28
```

Support Vector Machine

In [10]:
```
from sklearn.svm import LinearSVC

#Fitting SVM model and predicting the ratings
SVCmodel = LinearSVC(C = 0.5)
SVCmodel.fit(X_train_feature, Y_train)
prediction_svm = SVCmodel.predict(X_test_feature)

print("Accuracy of SVM Model using TF-IDF feature: 0.47 ")
svm_results = metrics.classification_report(Y_test, prediction_svm, output_dict=True)
accuracy = str(round(svm_results['accuracy'],2))

print("Accuracy of SVM Model using Word2Vec feature:", accuracy)
```

```
Accuracy of SVM Model using TF-IDF feature: 0.47
Accuracy of SVM Model using Word2Vec feature: 0.48
```

Conclusion and Answer to Q3

For simple models, I'm comparing the accuracy for Perceptron and Support Vector Machine trained on Tf-Idf model from homework 1 and same models trained with word2vec features from this homework. We can see the for perceptron, the accuracy was better for model trained with TF-IDF features but for Support Vector Machines, the accuracy was better with Word2Vec features than the Tf-Idf features. Since perceptron is a simple single layer neural network, the model needs more data to learn.

DataSet preparation and Dataloading

In [11]:
```
#Transforming labels into one-hot vector form for training the neural network models

Y_train = pd.get_dummies(Y_train)
Y_test = pd.get_dummies(Y_test)
```

In [12]:
```
from torch.utils.data import TensorDataset, DataLoader

# Transforming features and labels to tensors, forming a dataset using tensors and loa
# the dataset to DataLoader for training the Neural Network

train_feature_tensor = torch.tensor(X_train_feature, dtype=torch.float32)
test_feature_tensor = torch.tensor(X_test_feature, dtype=torch.float32)
train_label = torch.tensor(Y_train.values, dtype=torch.float32)
test_label = torch.tensor(Y_test.values, dtype=torch.float32)

train_data = TensorDataset(train_feature_tensor, train_label)
test_data = TensorDataset(test_feature_tensor, test_label)

batch_size = 50
train_loader = DataLoader(train_data, batch_size = batch_size, shuffle = True)
test_loader = DataLoader(test_data, batch_size = batch_size, shuffle = True)
```

```
C:\Users\ajeya\AppData\Local\Programs\Python\Python36\lib\site-packages\ipykernel_laun
cher.py:6: UserWarning: Creating a tensor from a list of numpy.ndarrays is extremely s
low. Please consider converting the list to a single numpy.ndarray with numpy.array()
before converting to a tensor. (Triggered internally at  ..\torch\csrc\utils\tensor_ne
w.cpp:201.)
```

Q4. Feedforward Neural Networks

Below class is for Feedforward neural network. It has two hidden layers of 50 and 10 nodes respectively. Input layer has 300 nodes to take 300-dim vector from word2vec feature as input and output layer has 5 nodes for 5 different classes for labels. I'm using cross entropy loss and SGD optimizer for training the neural network. Reference: https://www.kaggle.com/code/mishra1993/pytorch-multi-layer-perceptron-mnist/notebook

Q4-(a)

For this question, the feature vector is the weighted average of all the word2vec vector for each word in the review.

In [13]:
```
import torch.nn as nn
import torch.nn.functional as F

#This is the class for Feedforward neural network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(300, 50)
        self.fc2 = nn.Linear(50, 10)
        self.fc3 = nn.Linear(10, 5)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x = x.view(-1, 300)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x
        #return F.log_softmax(x, dim=0)

model = Net()
#print(model)


criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

n_epochs = 50

for epoch in range(n_epochs):
    train_loss = 0.0

    model.train()
    for data, target in train_loader:
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()*data.size(0)
    train_loss = train_loss/len(train_loader.dataset)
    #print(train_loss)
```

In below tab, I'm predicting the labels for reviews in testing set and calculating the accuracy of the neural network.

In [14]:
```
correct = 0
total = 0

with torch.no_grad():
    for data, target in test_loader:
        output = model(data)
        #print(output)
        for index, i in enumerate(output):
            if torch.argmax(i) == np.argmax(target[index]):
                correct += 1
            total += 1

#print(total)
#print(correct)
print("Q4-(a)Accuracy of Neural Network:", correct/total)
```

```
Q4-(a)Accuracy of Neural Network: 0.4786
```

Q4-(b)

In the below model,I'm changing the way feature is being generated. In previous question feature was generated using weighted average of all the word2vec vectors of the words in review. In this question, I'm concatenating the word2vec vector for first 10 words of the reviews (If review has less words, I'm concatenating zero vector for the feature. Due to this the input layer of the neural network should have 3000 nodes as our input feature is 3000-dim vector.

In [15]:
```
class NetConcat(nn.Module):
    def __init__(self):
        super(NetConcat, self).__init__()
        self.fc1 = nn.Linear(3000, 50)
        self.fc2 = nn.Linear(50, 10)
        self.fc3 = nn.Linear(10, 5)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x = x.view(-1, 3000)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x

def extractFeatureConcat(review):
    words = review.split(" ")
    words = words[0:10]
    sentence_vector = np.array([])
    for i in range(10):
        if(i < len(words) and words[i] in wv.key_to_index):
            word_vector = wv[words[i]]
            sentence_vector = np.concatenate([sentence_vector, word_vector])
        else:
            sentence_vector = np.concatenate([sentence_vector, np.zeros(300)])
    return sentence_vector

X_train_feature_concat = list(X_train.apply(extractFeatureConcat))
#print(len(X_train_feature_concat))
X_test_feature_concat = list(X_test.apply(extractFeatureConcat))
#print(len(X_test_feature_concat))
```

Below is the code for generating feature vector by concatenating word2vec vectors of first 10 words in the review.

In [16]:
```
train_feature_concat_tensor = torch.tensor(X_train_feature_concat, dtype=torch.float32
test_feature_concat_tensor = torch.tensor(X_test_feature_concat, dtype=torch.float32)
train_label = torch.tensor(Y_train.values, dtype=torch.float32)
test_label = torch.tensor(Y_test.values, dtype=torch.float32)

train_data_concat = TensorDataset(train_feature_concat_tensor, train_label)
test_data_concat = TensorDataset(test_feature_concat_tensor, test_label)

batch_size = 50
train_loader_concat = DataLoader(train_data_concat, batch_size = batch_size, shuffle =
test_loader_concat = DataLoader(test_data_concat, batch_size = batch_size, shuffle = T
```

In [17]:
```
model_concat = NetConcat()
#print(model_concat)

#Using CrossEntropyLoss for training and SGD for optimization
criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model_concat.parameters(), lr=0.01)

n_epochs = 50

#Training the model using the features generated.

for epoch in range(n_epochs):
    train_loss = 0.0

    model_concat.train()
    for data, target in train_loader_concat:
        optimizer.zero_grad()
        output = model_concat(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()*data.size(0)
    train_loss = train_loss/len(train_loader_concat.dataset)
    #print(train_loss)
```

Below is the code for predicting labels for reviews in testset using second neural network trained on feature vectors formed by concatenating the vectors of first 10 words.

In [18]:
```
correct = 0
total = 0

with torch.no_grad():
    for data, target in test_loader_concat:
        output = model_concat(data)
        for index, i in enumerate(output):
            if torch.argmax(i) == np.argmax(target[index]):
                correct += 1
            total += 1

#print(total)
#print(correct)
print("Q4-(b)Accuracy of Neural Network: :", correct/total)
```

```
Q4-(b)Accuracy of Neural Network: : 0.39815
```

Conclusion and answer to Q4

We see that the accuracy for feedforward neural networks is slightly lower than accuracy of SVM because nerual networks learns and performs well with more data. Since we had only 100,000 records, we could see a decent performance but we need more data to adjust right weights. We also see that neural networks performed better than the perceptron since neural networks have more layers and nodes than the perceptron which helps in learning better. Overall, more data would be useful for neural networks to learn.

Recurrent Neural Networks

Below is the class for recurrent neural network. For my model, as mentioned in Assignment description, I'm using hidden state size as 20, input layer size is 300 for 300-dim feature vector extracted from word2vec model. For each review, I use first feature vector of first 20 words to train the RNN model.

References:
https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html
https://pytorch.org/docs/stable/generated/torch.nn.GRU.html
https://pytorch.org/docs/stable/generated/torch.nn.RNN.html

Q5-(a)

For this question, class RNN is the recurrent neural network model I train and use for prediction.

In [19]:
```
#Preparing dataset for training RNN
X_train_rnn = X_train.values
Y_train_rnn = Y_train.values

X_test_rnn = X_test.values
Y_test_rnn = Y_test.values
```

Below is the code to form the dataset and dataloader for traing the RNN and gated RNN.

In [20]:
```
#Below function takes review as input and returns numpy array of feature vector as ou
def review_to_vector(review):
    words = review.split(" ")
    review_size = len(words)
    word_vector = np.zeros(shape=(20, 300))
    for i in range(0,20):
        if(review_size > i and words[i] in  wv.key_to_index):
            word_vector[i] = wv[words[i]]
    return word_vector

#Below is the code for forming feature dataset tensor for training the Gated RNN
Y_train_rnn = torch.tensor(Y_train_rnn, dtype=torch.float32)
Y_test_rnn = torch.tensor(Y_test_rnn, dtype=torch.float32)
X_train_rnn = np.array(list(map(review_to_vector, X_train_rnn)))
X_test_rnn = np.array(list(map(review_to_vector, X_test_rnn)))

X_train_rnn_tensor = torch.tensor(X_train_rnn, dtype=torch.float32)
X_test_rnn_tensor = torch.tensor(X_test_rnn, dtype=torch.float32)

#Form Tensor Dataset
train_data_rnn = TensorDataset(X_train_rnn_tensor, Y_train_rnn)
test_data_rnn = TensorDataset(X_test_rnn_tensor, Y_test_rnn)

#Load the dataset to DataLoader
batch_size = 50
train_loader_rnn = DataLoader(train_data_rnn, batch_size = batch_size, shuffle = True)
test_loader_rnn = DataLoader(test_data_rnn, batch_size = batch_size, shuffle = True)
```

In [21]:
```
#Class for RNN model
#Reference: https://pytorch.org/docs/stable/generated/torch.nn.RNN.html

class RNN(nn.Module):
    def __init__(self, layers, input_size, output_size):
        super(RNN, self).__init__()
        #RNN has 300 as input size to take 300-dim feature vector and 5 outputsize fo
        self.rnn = nn.RNN(input_size, 300, layers,batch_first=True)
        self.h0 = torch.randn(layers, 100, 300)
        self.linear = nn. Linear(300, output_size)


    def forward(self, input_tensor):
        x = self.rnn(input_tensor)[0][:,-1]
        #Since there are 5 classes, a linear layer is used to convert from 300 size to
        x = self.linear(x)
        return x

rnn_model = RNN(2, 300, 5)

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(rnn_model.parameters(), lr=0.01)

epochs = 5

for i in range(epochs):

    for data, target in train_loader_rnn:
        optimizer.zero_grad()
        output = rnn_model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()*data.size(0)
    train_loss = train_loss/len(train_loader_rnn.dataset)
    #print(train_loss)
```

In [22]:
```
correct = 0
total = 0

with torch.no_grad():
    for data, target in test_loader_rnn:
        output = rnn_model(data)
        for index, i in enumerate(output):
            if torch.argmax(i) == np.argmax(target[index]):
                correct += 1
            total += 1
#print(correct)
#print(total)
print("Q5-(a) Accuracy of RNN model:", correct/total)
```

```
Q5-(a) Accuracy of RNN model: 0.4351
```

Conclusion and answer to Q5-a

We see that the accuracy of RNN has decreased when compared to feedforward neural network. The reason can be found by understanding data better. RNN learns from the sequence of data, which is sequence of words here. Since there was less data, Feedforward neural network worked better than RNN, but with more data it might be possible that RNN learns better than the feedforward.

Q5-(b)

GRNN class is a class built for gated RNN. It has 300 nodes as input, 300 layers for hidden state and output size is 5 since we have 5 classes to predict.
Train Label Shape: [80000, 5]
Test Label Shape: [20000, 5]
Train Feature Shape: [80000, 20, 300]
Test Feature Shape: [20000, 20, 300]

In [23]:
```
#Below is the class for Gated RNN
#Reference: https://pytorch.org/docs/stable/generated/torch.nn.GRU.html

class GRNN(nn.Module):
    def __init__(self, layers, input_size, output_size):
        super(GRNN, self).__init__()
        #Input size is 300 to feed feature vectors, network has 2 layers and output s
        self.gru = nn.GRU(input_size, 300, layers,batch_first=True)
        self.h0 = torch.randn(layers, 100, 300)
        self.linear = nn. Linear(300, output_size)


    def forward(self, input_tensor):
        x = self.gru(input_tensor)[0][:,-1]
        #Output from the GRU is passed to next layer further get 5 output values
        x = self.linear(x)
        return x

gated_rnn = GRNN(2, 300, 5)

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(gated_rnn.parameters(), lr=0.01)
```

Below code form featureset used for training the Gated RNN model. For each review, a feature is formed by using first 20 words and converting it to word2vec feature vector

In [24]:
```
batch_size = 50
train_loader_gru = DataLoader(train_data_rnn, batch_size = batch_size, shuffle = True)
test_loader_gru = DataLoader(test_data_rnn, batch_size = batch_size, shuffle = True)

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(gated_rnn.parameters(), lr=0.01)

train_loss = 0
epochs = 5

#For loop for training the Gated RNN (GRNN) model
for i in range(epochs):

    for data, target in train_loader_gru:
        optimizer.zero_grad()
        output = gated_rnn(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()*data.size(0)

    train_loss = train_loss/len(train_loader_gru.dataset)
    #print(train_loss)
```

Below code is for predicting the labels for test dataset using the Gated RNN model. Accuracy is as shown below.

In [25]:
```
correct = 0
total = 0

#Below loop is used to calculate correct and total labels for gated RNN model to calcu
with torch.no_grad():
    for data, target in test_loader_gru:
        output = gated_rnn(data)
        for index, i in enumerate(output):
            if torch.argmax(i) == np.argmax(target[index]):
                correct += 1
            total += 1

print("Q5-(b) Accuracy of gated RNN model:", correct/total)
```

```
Q5-(b) Accuracy of gated RNN model: 0.3791
```

Conclusion and answer to Q5-b

There is not huge difference between accuracies of RNN and gated RNN. Gated RNN can sometimes remember better than the RNN for sequence data. But it is hard to analyze if gated RNN or RNN is better for this problem since review length varies with rows. As seen previously, the gated RNN model did not over perform feedforward NN for this problem with the data we have. For this problem RNN performed better than gated RNN.