In [1]:
```
#!/usr/bin/env python
# coding: utf-8
# # CSCI544_HW2_JingyanPeng
# - 09/26/2022

#version python3.9
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import nltk
import re
from bs4 import BeautifulSoup
from sklearn.metrics import accuracy_score
import torch
CUDA_LAUNCH_BLOCKING = "1"
```

In [3]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

Out[3]:
```
device(type='cuda')
```

1.Dataset Generation

- Build a Balanced Dataset Through Random Selection

  Load the dataset and build a balanced dataset of 100K reviews along with their ratings to create labels through random selection.

In [4]:
```
df = pd.read_csv('data.tsv', sep='\t', on_bad_lines='skip')
df['reviews'] = df['review_headline'] + ' ' + df['review_body']
df['ratings'] = df['star_rating']
df = df[['ratings','reviews']]
df = df.dropna()

s1=df[df.ratings == 1]
```

```
s2=df[df.ratings == 2]
s3=df[df.ratings == 3]
s4=df[df.ratings == 4]
s5=df[df.ratings == 5]
s1 = s1.sample(n = 20000, random_state = None)
s2 = s2.sample(n = 20000, random_state = None)
s3 = s3.sample(n = 20000, random_state = None)
s4 = s4.sample(n = 20000, random_state = None)
s5 = s5.sample(n = 20000, random_state = None)
dataset = pd.concat([s1, s2, s3, s4, s5])
dataset = dataset.reset_index(drop = True)
```

- Simple Data Cleaning without Preprocessing

In [5]:
```
dataset['reviews'] = dataset['reviews'].str.lower()

dataset['reviews'] = dataset['reviews'].map(lambda x: re.sub(re.compile(r'[http|https]*://[a-zA-Z0-9.?/&=:]*'
dataset['reviews'] = dataset['reviews'].map(lambda x: BeautifulSoup(x,"html.parser").get_text())

dataset['reviews'] = dataset['reviews'].map(lambda x: re.sub("[^a-zA-Z]+", " ", x))

dataset['reviews'] = dataset['reviews'].map(lambda x: re.sub(r'\s\s+', ' ', x))
#dataset['reviews'] = dataset['reviews'].map(lambda x: x.strip())

import contractions
def contractionFunc(s):
    s = contractions.fix(s)
    s = re.sub("[^a-zA-Z]+", " ", s)
    return s
dataset['reviews'] = dataset['reviews'].map(lambda x: contractionFunc(x))

print(dataset.head(5))
```

```
  ratings                                            reviews
0       1  really low quality crap product turned orange ...
1       1  waste of money unless you re buying them for a...
2       1  one star not really what i wanted it s way to ...
3       1  twice the price for half the amount this cost ...
4     1.0  can t believe it i didn t like anything about ...
```

2. Word Embedding

- Reference:
  - Gensim > Documentation > Word2Vec Model
  - https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html

- (a) Pretrained Word2Vec Model

Load the pretrained “word2vec-google-news-300” Word2Vec model.

In [6]:
```
import gensim.downloader as api
google_wv = api.load('word2vec-google-news-300')
```

Check semantic similarities of the generated vectors using wv.most_similar() & wv.similarity().

My own three examples:

In [7]:
```
print(google_wv.most_similar(positive=['uncle', 'woman'], negative=['man'], topn=1))
print(google_wv.similarity('bike', 'bicycle'))
print(google_wv.similarity('crucial', 'vital'))
```

```
[('aunt', 0.8022665977478027)]
0.85213083
0.82077205
```

Given two examples in assignment doc:

In [8]:
```
print(google_wv.most_similar(positive=['king', 'woman'], negative=['man'], topn=1))
print(google_wv.similarity('excellent', 'outstanding'))
```

```
[('queen', 0.7118193507194519)]
0.55674857
```

- (b) My Word2Vec Model

Train a Word2Vec model using my own dataset.

In [9]:
```
from gensim.test.utils import datapath
from gensim import utils
import gensim.models
```

```
class MyCorpus:
    def __iter__(self):
        for line in dataset['reviews']:
            yield utils.simple_preprocess(line)

model = gensim.models.Word2Vec(sentences=MyCorpus(), vector_size=300, window=11, min_count=10)
```

Check semantic similarities of the generated vectors using wv.most_similar() & wv.similarity().

Given two examples in assignment doc:

In [10]:
```
print(model.wv.most_similar(positive=['king', 'woman'], negative=['man'], topn=1))
print(model.wv.similarity('excellent', 'outstanding'))
```

```
[('poem', 0.5422888994216919)]
0.81494856
```

- Conclusion
  - It shows that for the example of 'King – Man + Woman = Queen', the “word2vec-google-news-300” Word2Vec model works better. The model trained with my own dataset cannot give the answer, 'queen'. But for the example of 'excellent ~ outstanding'. The model trained with my own dataset works better (0.81 > 0.56).
  - This may be related to characteristics of different datasets. If the words appears often in the dataset, as 'excellent' and 'outstanding' is more commonly used in my own dataset than in google news dataset, the corresponding vectors of these commonly used words trained with this kind of dataset can be more accurate.

Train / Test split

80%/20% training/testing

In [11]:
```
train = dataset.sample(frac = 0.8, random_state = 1)
test = dataset.drop(train.index)
train = train.reset_index(drop = True)
test = test.reset_index(drop = True)

X_train = train['reviews']
```

```
X_test = test['reviews']
Y_train = train['ratings']
Y_test = test['ratings']
```

Function Definition for Word2Vec -> Input

Here I define 5 functions to process the Word2Vec to input data.

1. Delete the corresponding labels of the NaN training vectors.
2. Delete the NaN training vectors.
3. Calculate the average vector for each review.
4. Concatenate the first 10 vectors for each review. Truncate the longer one and pad the shorter one with 0.
5. VStack the first 20 vectors for each review. Truncate the longer one and pad the shorter one with 0.

In [12]:
```
# 1) processing NaN
def process_nanY(x_mtx, y_mtx):
    idx = []
    if np.any(np.isnan(x_mtx)):
        arr_nan = np.argwhere(np.isnan(x_mtx))
        num_nan = arr_nan.shape[0]
        arr = np.arange(0, num_nan, 300)
        for i in arr:
            idx.append(arr_nan[i][0])
    if idx != None:
        mtx = np.delete(y_mtx, idx)
    return mtx;

def process_nanX(x_mtx):
    idx = []
    if np.any(np.isnan(x_mtx)):
        arr_nan = np.argwhere(np.isnan(x_mtx))
        num_nan = arr_nan.shape[0]
        arr = np.arange(0, num_nan, 300)
        for i in arr:
            idx.append(arr_nan[i][0])
```

```
    if idx != None:
        mtx = np.delete(x_mtx, idx, 0)
    return mtx;


# 2) the average Word2Vec vectors
def w2v_average(wv_model, input_words):
    wordlist = input_words.split(' ')
    embed_sum = np.zeros(shape = (300,))
    count = 0
    for word in wordlist:
        if word in wv_model:
            embed_sum += wv_model[word]
            count += 1
    return embed_sum / count

# 3) concatenate the first 10 Word2Vec vectors
def w2v_first10(wv_model, input_words):
    wordlist = input_words.split(' ')
    idx = 0;
    coun = 0;
    # go through the reviews to find 10 words in W2V model
    while(idx < len(wordlist)) & (coun <10):
        if wordlist[idx] in wv_model:
            wv_current = wv_model[wordlist[idx]]
            if coun == 0:
                result = wv_current
            else:
                result = np.concatenate((result, wv_current))
            idx += 1
            coun += 1
        else:
            idx +=1
    # if reviews length < 10:
    if coun == 0:
        return np.zeros(shape = 3000, )
    if coun < 10:
        zeros = np.zeros(shape = (300 * (10 - coun), ))
        return np.concatenate((result, zeros))
    else:
        return result

# 4) vStack the first 20 Word2Vec vectors
```

```
#    limit the review length to 20 by truncating and padding
def w2v_seq20(wv_model, input_words):
    wordlist = input_words.split(' ')
    idx = 0;
    coun = 0;
    # go through the reviews to find 10 words in W2V model
    while(idx < len(wordlist)) & (coun < 20):
        if wordlist[idx] in wv_model:
            wv_current = wv_model[wordlist[idx]]
            if coun == 0:
                result = wv_current
            else:
                result = np.vstack((result, wv_current))
            idx += 1
            coun += 1
        else:
            idx +=1
    # if reviews length < 20:
    if coun == 0:
        return np.zeros(shape = (20, 300) )
    if coun < 20:
        zeros = np.zeros(shape = (20 - coun, 300))
        return np.vstack((result, zeros))
    else:
        return result
```

3. Simple Model

In [13]:
```
#Calculate the average Word2Vec vectors of Google-pretrained Word2Vec Model.
X_train_preW2Vave =np.array(X_train.apply(lambda x: w2v_average(google_wv, x)).values.tolist())
X_test_preW2Vave =np.array(X_test.apply(lambda x: w2v_average(google_wv, x)).values.tolist())
Y_train_preW2Vave = np.array(Y_train.values.tolist())
Y_test_preW2Vave = np.array(Y_test.values.tolist())
#Delete NaN Vectors with their Labels.
Y_train_preW2Vave = process_nanY(X_train_preW2Vave, Y_train_preW2Vave)
X_train_preW2Vave = process_nanX(X_train_preW2Vave)
Y_test_preW2Vave = process_nanY(X_test_preW2Vave, Y_test_preW2Vave)
X_test_preW2Vave = process_nanX(X_test_preW2Vave)
## print(X_train_preW2Vave.shape)
```

- Perceptron

Use sklearn.linear_model.Perceptron().fit() to train a Perceptron model and sklearn.linear_model.Perceptron().predict() to get the accurancy.

In [14]:
```
from sklearn.linear_model import Perceptron
perceptron_pre = Perceptron(max_iter = 1000, tol = 0, random_state = 0, eta0 = 0.01)
perceptron_pre.fit(X_train_preW2Vave, Y_train_preW2Vave)
perceptron_pre_test = perceptron_pre.predict(X_test_preW2Vave)
perceptron_pre_test_accuracy = accuracy_score(Y_test_preW2Vave, perceptron_pre_test)
print(perceptron_pre_test_accuracy)
```

```
0.4594959495949595
```

- SVM

Use sklearn.svm.LinearSVC().fit() to train a SVM model and sklearn.svm.LinearSVC().predict() to get the accurancy.

In [15]:
```
from sklearn.svm import LinearSVC
svm_pre = LinearSVC(max_iter = 5000, random_state = 0)
svm_pre.fit(X_train_preW2Vave, Y_train_preW2Vave)
svm_pre_test = svm_pre.predict(X_test_preW2Vave)
svm_pre_test_accurancy = accuracy_score(Y_test_preW2Vave, svm_pre_test)
print(svm_pre_test_accurancy)
```

```
0.5621562156215621
```

- Conclusion
  - In HW1, I got the accurancy values of these simple models with TF-IDF as input——0.42 for Perceptron model and 0.44 for SVM model. They are all worse than the results using pretrained Word2Vec as input here——0.46 for Perceptron model and 0.56 for SVM model.
  - So, in this case, the Word2Vec model works better than TF-IDF for words embedding.

Custom Dataset

Define the dataset for Neural Network training.

In [16]:
```
import torch
from torch.utils.data import DataLoader, Dataset
from torch.utils.data.sampler import SubsetRandomSampler
import torch.nn as nn
import torch.nn.functional as F
import time
class Train(Dataset):
    def __init__(self, xtrain, ytrain):
        self.data = xtrain
        self.labels = ytrain

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        X = self.data[index]
        y = self.labels[index]
        return X, y


class Test(Dataset):
    def __init__(self, xtest, ytest):
        self.data = xtest
        self.labels = ytest

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        X = self.data[index]
        y = self.labels[index]
        return X, y
```

4. Feedforward Neural Networks

- Reference:
  - Pytorch Multi-Layer Perceptron, MNIST | Author: BHARAT BUSHAN MISHRA
  - https://www.kaggle.com/code/mishra1993/pytorch-multi-layer-perceptron-mnist/notebook

Define the Network Architecture (MLP)

In [17]:
```
# define the NN architecture
class MLP(nn.Module):
    def __init__(self, D_input, D_output):
        super(MLP, self).__init__()
        # number of hidden nodes in each layer
        # layer1: 50 nodes; layer2: 10 nodes
        hidden_1 = 50
        hidden_2 = 10
        # linear layer (300 -> hidden_1)
        self.fc1 = nn.Linear(D_input, hidden_1)
        # linear layer (n_hidden -> hidden_2)
        self.fc2 = nn.Linear(hidden_1, hidden_2)
        # linear layer (n_hidden -> 5)
        self.fc3 = nn.Linear(hidden_2, D_output)
        # dropout layer (p=0.2)
        # dropout prevents overfitting of data
        self.dropout = nn.Dropout(0.2)
    def forward(self, x):
        # add hidden layer, with relu activation function
        x = F.relu(self.fc1(x))
        # add dropout layer
        x = self.dropout(x)
        # add hidden layer, with relu activation function
        x = F.relu(self.fc2(x))
        # add dropout layer
        x = self.dropout(x)
        # add output layer
        x = self.fc3(x)
        return x

# initialize the NN
model_4a = MLP(300, 5)
model_4b = MLP(3000, 5)
model_4a.cuda()
model_4b.cuda()
print(model_4a)
print(model_4b)

# specify loss function (categorical cross-entropy)
criterion = nn.CrossEntropyLoss()
# specify optimizer (stochastic gradient descent) and learning rate = 0.01
```

```
optimizer_4a = torch.optim.SGD(model_4a.parameters(), lr=0.01)
optimizer_4b = torch.optim.SGD(model_4b.parameters(), lr=0.01)
```

```
MLP(
  (fc1): Linear(in_features=300, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=10, bias=True)
  (fc3): Linear(in_features=10, out_features=5, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
MLP(
  (fc1): Linear(in_features=3000, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=10, bias=True)
  (fc3): Linear(in_features=10, out_features=5, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

- (a) the average Word2Vec vectors as input

In [18]:
```
############################## Load the data_4a ##############################
train_data_4a = Train(X_train_preW2Vave, Y_train_preW2Vave-1)
test_data_4a = Test(X_test_preW2Vave, Y_test_preW2Vave-1)

# number of subprocesses to use for data loading
num_workers = 0
# how many samples per batch to load
batch_size = 100
# percentage of training set to use as validation
valid_size = 0.2

# obtain training indices that will be used for validation
num_train = len(train_data_4a)
indices = list(range(num_train))
np.random.shuffle(indices)
split = int(np.floor(valid_size * num_train))
train_idx, valid_idx = indices[split:], indices[:split]

# define samplers for obtaining training and validation batches
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

# prepare data loaders
train_loader_4a = torch.utils.data.DataLoader(train_data_4a, batch_size=batch_size, sampler=train_sampler, nu
```

```
valid_loader_4a = torch.utils.data.DataLoader(train_data_4a, batch_size=batch_size, sampler=valid_sampler, nu
test_loader_4a = torch.utils.data.DataLoader(test_data_4a, batch_size=batch_size, num_workers=num_workers)
```

In [19]:

```
############################## Train the model_4a ##############################
start = time.time()

# number of epochs to train the model
n_epochs = 150

# initialize tracker for minimum validation loss
valid_loss_min = np.Inf # set initial "min" to infinity

for epoch in range(n_epochs):
    # monitor training loss
    train_loss = 0.0
    valid_loss = 0.0

    ###################
    # train the model #
    ###################
    model_4a.train() # prep model for training
    for data, target in train_loader_4a:
        # transfer data and target to GPU
        data, target = data.to(device), target.to(device)
        # clear the gradients of all optimized variables
        optimizer_4a.zero_grad()
        # forward pass: compute predicted outputs by passing inputs to the model
        output = model_4a(data.float())
        # calculate the loss
        loss = criterion(output, target.to(torch.long))
        # backward pass: compute gradient of the loss with respect to model parameters
        loss.backward()
        # perform a single optimization step (parameter update)
        optimizer_4a.step()
        # update running training loss
        train_loss += loss.item()*data.size(0)

    ######################
    # validate the model #
    ######################
    model_4a.eval() # prep model for evaluation
    for data, target in valid_loader_4a:
        # transfer data and target to GPU
```

```
        data, target = data.to(device), target.to(device)
        # forward pass: compute predicted outputs by passing inputs to the model
        output = model_4a(data.float())
        # calculate the loss
        loss = criterion(output, target.to(torch.long))
        # update running validation loss
        valid_loss += loss.item()*data.size(0)

    # print training/validation statistics
    # calculate average loss over an epoch
    train_loss = train_loss/len(train_loader_4a.dataset)
    valid_loss = valid_loss/len(valid_loader_4a.dataset)

    print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
        epoch+1,
        train_loss,
        valid_loss
        ))

    # save model if validation loss has decreased
    if valid_loss <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(
        valid_loss_min,
        valid_loss))
        torch.save(model_4a.state_dict(), 'model.pt')
        valid_loss_min = valid_loss

end = time.time()
print('Time elapsed: %.2f s' % (end - start))
```

```
Epoch: 1        Training Loss: 1.291888         Validation Loss: 0.322190
Validation loss decreased (inf --> 0.322190).  Saving model ...
Epoch: 2        Training Loss: 1.287869         Validation Loss: 0.321915
Validation loss decreased (0.322190 --> 0.321915).  Saving model ...
Epoch: 3        Training Loss: 1.287442         Validation Loss: 0.321847
Validation loss decreased (0.321915 --> 0.321847).  Saving model ...
Epoch: 4        Training Loss: 1.287303         Validation Loss: 0.321805
Validation loss decreased (0.321847 --> 0.321805).  Saving model ...
Epoch: 5        Training Loss: 1.287180         Validation Loss: 0.321782
Validation loss decreased (0.321805 --> 0.321782).  Saving model ...
Epoch: 6        Training Loss: 1.287018         Validation Loss: 0.321735
Validation loss decreased (0.321782 --> 0.321735).  Saving model ...
Epoch: 7        Training Loss: 1.286869         Validation Loss: 0.321695
Validation loss decreased (0.321735 --> 0.321695).  Saving model ...
Epoch: 8        Training Loss: 1.286668         Validation Loss: 0.321644
Validation loss decreased (0.321695 --> 0.321644).  Saving model ...
Epoch: 9        Training Loss: 1.286455         Validation Loss: 0.321586
Validation loss decreased (0.321644 --> 0.321586).  Saving model ...
Epoch: 10       Training Loss: 1.286206         Validation Loss: 0.321513
Validation loss decreased (0.321586 --> 0.321513).  Saving model ...
Epoch: 11       Training Loss: 1.285757         Validation Loss: 0.321406
Validation loss decreased (0.321513 --> 0.321406).  Saving model ...
Epoch: 12       Training Loss: 1.285391         Validation Loss: 0.321282
Validation loss decreased (0.321406 --> 0.321282).  Saving model ...
Epoch: 13       Training Loss: 1.284819         Validation Loss: 0.321123
Validation loss decreased (0.321282 --> 0.321123).  Saving model ...
Epoch: 14       Training Loss: 1.284045         Validation Loss: 0.320900
Validation loss decreased (0.321123 --> 0.320900).  Saving model ...
Epoch: 15       Training Loss: 1.283021         Validation Loss: 0.320568
Validation loss decreased (0.320900 --> 0.320568).  Saving model ...
Epoch: 16       Training Loss: 1.281132         Validation Loss: 0.319953
Validation loss decreased (0.320568 --> 0.319953).  Saving model ...
Epoch: 17       Training Loss: 1.278506         Validation Loss: 0.319193
Validation loss decreased (0.319953 --> 0.319193).  Saving model ...
Epoch: 18       Training Loss: 1.274754         Validation Loss: 0.318010
Validation loss decreased (0.319193 --> 0.318010).  Saving model ...
Epoch: 19       Training Loss: 1.269149         Validation Loss: 0.315945
Validation loss decreased (0.318010 --> 0.315945).  Saving model ...
Epoch: 20       Training Loss: 1.259047         Validation Loss: 0.312754
Validation loss decreased (0.315945 --> 0.312754).  Saving model ...
Epoch: 21       Training Loss: 1.244182         Validation Loss: 0.307929
Validation loss decreased (0.312754 --> 0.307929).  Saving model ...
Epoch: 22       Training Loss: 1.224290         Validation Loss: 0.301651
```

```
Validation loss decreased (0.307929 --> 0.301651).  Saving model ...
Epoch: 23       Training Loss: 1.200268         Validation Loss: 0.294882
Validation loss decreased (0.301651 --> 0.294882).  Saving model ...
Epoch: 24       Training Loss: 1.177957         Validation Loss: 0.288566
Validation loss decreased (0.294882 --> 0.288566).  Saving model ...
Epoch: 25       Training Loss: 1.156982         Validation Loss: 0.282964
Validation loss decreased (0.288566 --> 0.282964).  Saving model ...
Epoch: 26       Training Loss: 1.137607         Validation Loss: 0.277914
Validation loss decreased (0.282964 --> 0.277914).  Saving model ...
Epoch: 27       Training Loss: 1.120865         Validation Loss: 0.273507
Validation loss decreased (0.277914 --> 0.273507).  Saving model ...
Epoch: 28       Training Loss: 1.106743         Validation Loss: 0.269681
Validation loss decreased (0.273507 --> 0.269681).  Saving model ...
Epoch: 29       Training Loss: 1.094160         Validation Loss: 0.266199
Validation loss decreased (0.269681 --> 0.266199).  Saving model ...
Epoch: 30       Training Loss: 1.080650         Validation Loss: 0.263057
Validation loss decreased (0.266199 --> 0.263057).  Saving model ...
Epoch: 31       Training Loss: 1.072850         Validation Loss: 0.260479
Validation loss decreased (0.263057 --> 0.260479).  Saving model ...
Epoch: 32       Training Loss: 1.061878         Validation Loss: 0.258121
Validation loss decreased (0.260479 --> 0.258121).  Saving model ...
Epoch: 33       Training Loss: 1.053866         Validation Loss: 0.255661
Validation loss decreased (0.258121 --> 0.255661).  Saving model ...
Epoch: 34       Training Loss: 1.044948         Validation Loss: 0.253541
Validation loss decreased (0.255661 --> 0.253541).  Saving model ...
Epoch: 35       Training Loss: 1.037253         Validation Loss: 0.251743
Validation loss decreased (0.253541 --> 0.251743).  Saving model ...
Epoch: 36       Training Loss: 1.030884         Validation Loss: 0.250129
Validation loss decreased (0.251743 --> 0.250129).  Saving model ...
Epoch: 37       Training Loss: 1.023242         Validation Loss: 0.248392
Validation loss decreased (0.250129 --> 0.248392).  Saving model ...
Epoch: 38       Training Loss: 1.019616         Validation Loss: 0.246983
Validation loss decreased (0.248392 --> 0.246983).  Saving model ...
Epoch: 39       Training Loss: 1.013652         Validation Loss: 0.245626
Validation loss decreased (0.246983 --> 0.245626).  Saving model ...
Epoch: 40       Training Loss: 1.008268         Validation Loss: 0.244319
Validation loss decreased (0.245626 --> 0.244319).  Saving model ...
Epoch: 41       Training Loss: 1.004315         Validation Loss: 0.243117
Validation loss decreased (0.244319 --> 0.243117).  Saving model ...
Epoch: 42       Training Loss: 1.001012         Validation Loss: 0.241775
Validation loss decreased (0.243117 --> 0.241775).  Saving model ...
Epoch: 43       Training Loss: 0.994878         Validation Loss: 0.240578
Validation loss decreased (0.241775 --> 0.240578).  Saving model ...
```

```
Epoch: 44       Training Loss: 0.991673         Validation Loss: 0.239592
Validation loss decreased (0.240578 --> 0.239592).  Saving model ...
Epoch: 45       Training Loss: 0.986597         Validation Loss: 0.238503
Validation loss decreased (0.239592 --> 0.238503).  Saving model ...
Epoch: 46       Training Loss: 0.984068         Validation Loss: 0.237756
Validation loss decreased (0.238503 --> 0.237756).  Saving model ...
Epoch: 47       Training Loss: 0.979601         Validation Loss: 0.236679
Validation loss decreased (0.237756 --> 0.236679).  Saving model ...
Epoch: 48       Training Loss: 0.976399         Validation Loss: 0.235866
Validation loss decreased (0.236679 --> 0.235866).  Saving model ...
Epoch: 49       Training Loss: 0.973594         Validation Loss: 0.235080
Validation loss decreased (0.235866 --> 0.235080).  Saving model ...
Epoch: 50       Training Loss: 0.969694         Validation Loss: 0.234169
Validation loss decreased (0.235080 --> 0.234169).  Saving model ...
Epoch: 51       Training Loss: 0.968368         Validation Loss: 0.233687
Validation loss decreased (0.234169 --> 0.233687).  Saving model ...
Epoch: 52       Training Loss: 0.964001         Validation Loss: 0.232724
Validation loss decreased (0.233687 --> 0.232724).  Saving model ...
Epoch: 53       Training Loss: 0.962887         Validation Loss: 0.232101
Validation loss decreased (0.232724 --> 0.232101).  Saving model ...
Epoch: 54       Training Loss: 0.958259         Validation Loss: 0.231742
Validation loss decreased (0.232101 --> 0.231742).  Saving model ...
Epoch: 55       Training Loss: 0.958435         Validation Loss: 0.230670
Validation loss decreased (0.231742 --> 0.230670).  Saving model ...
Epoch: 56       Training Loss: 0.953616         Validation Loss: 0.230265
Validation loss decreased (0.230670 --> 0.230265).  Saving model ...
Epoch: 57       Training Loss: 0.951956         Validation Loss: 0.229575
Validation loss decreased (0.230265 --> 0.229575).  Saving model ...
Epoch: 58       Training Loss: 0.949562         Validation Loss: 0.228891
Validation loss decreased (0.229575 --> 0.228891).  Saving model ...
Epoch: 59       Training Loss: 0.947658         Validation Loss: 0.228967
Epoch: 60       Training Loss: 0.944441         Validation Loss: 0.228231
Validation loss decreased (0.228891 --> 0.228231).  Saving model ...
Epoch: 61       Training Loss: 0.943739         Validation Loss: 0.227487
Validation loss decreased (0.228231 --> 0.227487).  Saving model ...
Epoch: 62       Training Loss: 0.940192         Validation Loss: 0.226727
Validation loss decreased (0.227487 --> 0.226727).  Saving model ...
Epoch: 63       Training Loss: 0.939797         Validation Loss: 0.226221
Validation loss decreased (0.226727 --> 0.226221).  Saving model ...
Epoch: 64       Training Loss: 0.937761         Validation Loss: 0.225837
Validation loss decreased (0.226221 --> 0.225837).  Saving model ...
Epoch: 65       Training Loss: 0.934809         Validation Loss: 0.225310
Validation loss decreased (0.225837 --> 0.225310).  Saving model ...
```

```
Epoch: 66       Training Loss: 0.934270         Validation Loss: 0.224736
Validation loss decreased (0.225310 --> 0.224736).  Saving model ...
Epoch: 67       Training Loss: 0.932015         Validation Loss: 0.224418
Validation loss decreased (0.224736 --> 0.224418).  Saving model ...
Epoch: 68       Training Loss: 0.928870         Validation Loss: 0.223870
Validation loss decreased (0.224418 --> 0.223870).  Saving model ...
Epoch: 69       Training Loss: 0.926700         Validation Loss: 0.223305
Validation loss decreased (0.223870 --> 0.223305).  Saving model ...
Epoch: 70       Training Loss: 0.924372         Validation Loss: 0.222808
Validation loss decreased (0.223305 --> 0.222808).  Saving model ...
Epoch: 71       Training Loss: 0.923455         Validation Loss: 0.222419
Validation loss decreased (0.222808 --> 0.222419).  Saving model ...
Epoch: 72       Training Loss: 0.922550         Validation Loss: 0.221885
Validation loss decreased (0.222419 --> 0.221885).  Saving model ...
Epoch: 73       Training Loss: 0.919910         Validation Loss: 0.221446
Validation loss decreased (0.221885 --> 0.221446).  Saving model ...
Epoch: 74       Training Loss: 0.918084         Validation Loss: 0.221226
Validation loss decreased (0.221446 --> 0.221226).  Saving model ...
Epoch: 75       Training Loss: 0.915768         Validation Loss: 0.221101
Validation loss decreased (0.221226 --> 0.221101).  Saving model ...
Epoch: 76       Training Loss: 0.914186         Validation Loss: 0.220374
Validation loss decreased (0.221101 --> 0.220374).  Saving model ...
Epoch: 77       Training Loss: 0.913734         Validation Loss: 0.219936
Validation loss decreased (0.220374 --> 0.219936).  Saving model ...
Epoch: 78       Training Loss: 0.910765         Validation Loss: 0.219452
Validation loss decreased (0.219936 --> 0.219452).  Saving model ...
Epoch: 79       Training Loss: 0.909278         Validation Loss: 0.219036
Validation loss decreased (0.219452 --> 0.219036).  Saving model ...
Epoch: 80       Training Loss: 0.907350         Validation Loss: 0.218699
Validation loss decreased (0.219036 --> 0.218699).  Saving model ...
Epoch: 81       Training Loss: 0.906042         Validation Loss: 0.218240
Validation loss decreased (0.218699 --> 0.218240).  Saving model ...
Epoch: 82       Training Loss: 0.903571         Validation Loss: 0.217489
Validation loss decreased (0.218240 --> 0.217489).  Saving model ...
Epoch: 83       Training Loss: 0.901780         Validation Loss: 0.217059
Validation loss decreased (0.217489 --> 0.217059).  Saving model ...
Epoch: 84       Training Loss: 0.900054         Validation Loss: 0.216852
Validation loss decreased (0.217059 --> 0.216852).  Saving model ...
Epoch: 85       Training Loss: 0.900946         Validation Loss: 0.217625
Epoch: 86       Training Loss: 0.898363         Validation Loss: 0.216348
Validation loss decreased (0.216852 --> 0.216348).  Saving model ...
Epoch: 87       Training Loss: 0.896153         Validation Loss: 0.215803
Validation loss decreased (0.216348 --> 0.215803).  Saving model ...
```

```
Epoch: 88       Training Loss: 0.895205         Validation Loss: 0.215502
Validation loss decreased (0.215803 --> 0.215502).  Saving model ...
Epoch: 89       Training Loss: 0.893839         Validation Loss: 0.214995
Validation loss decreased (0.215502 --> 0.214995).  Saving model ...
Epoch: 90       Training Loss: 0.894365         Validation Loss: 0.215363
Epoch: 91       Training Loss: 0.891835         Validation Loss: 0.214529
Validation loss decreased (0.214995 --> 0.214529).  Saving model ...
Epoch: 92       Training Loss: 0.890510         Validation Loss: 0.214373
Validation loss decreased (0.214529 --> 0.214373).  Saving model ...
Epoch: 93       Training Loss: 0.887154         Validation Loss: 0.214060
Validation loss decreased (0.214373 --> 0.214060).  Saving model ...
Epoch: 94       Training Loss: 0.888484         Validation Loss: 0.213805
Validation loss decreased (0.214060 --> 0.213805).  Saving model ...
Epoch: 95       Training Loss: 0.887158         Validation Loss: 0.213733
Validation loss decreased (0.213805 --> 0.213733).  Saving model ...
Epoch: 96       Training Loss: 0.885518         Validation Loss: 0.213589
Validation loss decreased (0.213733 --> 0.213589).  Saving model ...
Epoch: 97       Training Loss: 0.883553         Validation Loss: 0.212451
Validation loss decreased (0.213589 --> 0.212451).  Saving model ...
Epoch: 98       Training Loss: 0.882370         Validation Loss: 0.212272
Validation loss decreased (0.212451 --> 0.212272).  Saving model ...
Epoch: 99       Training Loss: 0.881347         Validation Loss: 0.212382
Epoch: 100      Training Loss: 0.880502         Validation Loss: 0.212003
Validation loss decreased (0.212272 --> 0.212003).  Saving model ...
Epoch: 101      Training Loss: 0.879786         Validation Loss: 0.211565
Validation loss decreased (0.212003 --> 0.211565).  Saving model ...
Epoch: 102      Training Loss: 0.877767         Validation Loss: 0.211241
Validation loss decreased (0.211565 --> 0.211241).  Saving model ...
Epoch: 103      Training Loss: 0.874903         Validation Loss: 0.210906
Validation loss decreased (0.211241 --> 0.210906).  Saving model ...
Epoch: 104      Training Loss: 0.877778         Validation Loss: 0.211165
Epoch: 105      Training Loss: 0.875874         Validation Loss: 0.211264
Epoch: 106      Training Loss: 0.875490         Validation Loss: 0.210402
Validation loss decreased (0.210906 --> 0.210402).  Saving model ...
Epoch: 107      Training Loss: 0.874646         Validation Loss: 0.210697
Epoch: 108      Training Loss: 0.872720         Validation Loss: 0.210000
Validation loss decreased (0.210402 --> 0.210000).  Saving model ...
Epoch: 109      Training Loss: 0.872451         Validation Loss: 0.209605
Validation loss decreased (0.210000 --> 0.209605).  Saving model ...
Epoch: 110      Training Loss: 0.870960         Validation Loss: 0.209950
Epoch: 111      Training Loss: 0.869272         Validation Loss: 0.209404
Validation loss decreased (0.209605 --> 0.209404).  Saving model ...
Epoch: 112      Training Loss: 0.869148         Validation Loss: 0.208757
```

```
Validation loss decreased (0.209404 --> 0.208757).  Saving model ...
Epoch: 113      Training Loss: 0.867739         Validation Loss: 0.209077
Epoch: 114      Training Loss: 0.865680         Validation Loss: 0.208785
Epoch: 115      Training Loss: 0.864672         Validation Loss: 0.209393
Epoch: 116      Training Loss: 0.863473         Validation Loss: 0.208004
Validation loss decreased (0.208757 --> 0.208004).  Saving model ...
Epoch: 117      Training Loss: 0.863373         Validation Loss: 0.208034
Epoch: 118      Training Loss: 0.862995         Validation Loss: 0.207768
Validation loss decreased (0.208004 --> 0.207768).  Saving model ...
Epoch: 119      Training Loss: 0.862538         Validation Loss: 0.208044
Epoch: 120      Training Loss: 0.862091         Validation Loss: 0.207252
Validation loss decreased (0.207768 --> 0.207252).  Saving model ...
Epoch: 121      Training Loss: 0.860212         Validation Loss: 0.207692
Epoch: 122      Training Loss: 0.859222         Validation Loss: 0.206571
Validation loss decreased (0.207252 --> 0.206571).  Saving model ...
Epoch: 123      Training Loss: 0.859297         Validation Loss: 0.207450
Epoch: 124      Training Loss: 0.856474         Validation Loss: 0.207396
Epoch: 125      Training Loss: 0.857119         Validation Loss: 0.206671
Epoch: 126      Training Loss: 0.857831         Validation Loss: 0.206400
Validation loss decreased (0.206571 --> 0.206400).  Saving model ...
Epoch: 127      Training Loss: 0.854012         Validation Loss: 0.206006
Validation loss decreased (0.206400 --> 0.206006).  Saving model ...
Epoch: 128      Training Loss: 0.853439         Validation Loss: 0.205778
Validation loss decreased (0.206006 --> 0.205778).  Saving model ...
Epoch: 129      Training Loss: 0.855424         Validation Loss: 0.205610
Validation loss decreased (0.205778 --> 0.205610).  Saving model ...
Epoch: 130      Training Loss: 0.851856         Validation Loss: 0.205756
Epoch: 131      Training Loss: 0.851434         Validation Loss: 0.205709
Epoch: 132      Training Loss: 0.850421         Validation Loss: 0.204863
Validation loss decreased (0.205610 --> 0.204863).  Saving model ...
Epoch: 133      Training Loss: 0.849895         Validation Loss: 0.205686
Epoch: 134      Training Loss: 0.851511         Validation Loss: 0.204852
Validation loss decreased (0.204863 --> 0.204852).  Saving model ...
Epoch: 135      Training Loss: 0.847761         Validation Loss: 0.204418
Validation loss decreased (0.204852 --> 0.204418).  Saving model ...
Epoch: 136      Training Loss: 0.847176         Validation Loss: 0.204241
Validation loss decreased (0.204418 --> 0.204241).  Saving model ...
Epoch: 137      Training Loss: 0.846153         Validation Loss: 0.206336
Epoch: 138      Training Loss: 0.846038         Validation Loss: 0.204056
Validation loss decreased (0.204241 --> 0.204056).  Saving model ...
Epoch: 139      Training Loss: 0.846941         Validation Loss: 0.203407
Validation loss decreased (0.204056 --> 0.203407).  Saving model ...
Epoch: 140      Training Loss: 0.844332         Validation Loss: 0.204351
```

```
Epoch: 141      Training Loss: 0.844347         Validation Loss: 0.203626
Epoch: 142      Training Loss: 0.842002         Validation Loss: 0.203224
Validation loss decreased (0.203407 --> 0.203224).  Saving model ...
Epoch: 143      Training Loss: 0.842229         Validation Loss: 0.202874
Validation loss decreased (0.203224 --> 0.202874).  Saving model ...
Epoch: 144      Training Loss: 0.840950         Validation Loss: 0.202543
Validation loss decreased (0.202874 --> 0.202543).  Saving model ...
Epoch: 145      Training Loss: 0.842863         Validation Loss: 0.202679
Epoch: 146      Training Loss: 0.843186         Validation Loss: 0.203025
Epoch: 147      Training Loss: 0.839189         Validation Loss: 0.202223
Validation loss decreased (0.202543 --> 0.202223).  Saving model ...
Epoch: 148      Training Loss: 0.839394         Validation Loss: 0.202036
Validation loss decreased (0.202223 --> 0.202036).  Saving model ...
Epoch: 149      Training Loss: 0.837752         Validation Loss: 0.202530
Epoch: 150      Training Loss: 0.838405         Validation Loss: 0.202111
Time elapsed: 228.71 s
```

In [20]:
```
##############################  Calculate & Print the Accurancy_4a on Test#################################
# Load the model with the lowest validation loss
model_4a.load_state_dict(torch.load('model.pt'))
# Calculate the accurancy
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader_4a:
        embeddings, labels = data
        # calculating outputs by running embeddings through the network
        model_4a.to("cpu")
        outputs = model_4a(embeddings.float())
        # the class with the highest score is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(correct/total)
```

```
0.5713071307130713
```

- (b) concatenate the first 10 Word2Vec vectors as input

In [21]:
```
#Concatenate the first 10 Word2Vec vectors of Google-pretrained Word2Vec Model.
X_train_preW2Vfirst10 =np.array(X_train.apply(lambda x: w2v_first10(google_wv, x)).values.tolist())
```

```
X_test_preW2Vfirst10 =np.array(X_test.apply(lambda x: w2v_first10(google_wv, x)).values.tolist())
Y_train_preW2Vfirst10 = np.array(Y_train.values.tolist())
Y_test_preW2Vfirst10 = np.array(Y_test.values.tolist())
#Delete NaN Vectors with their Labels
Y_train_preW2Vfirst10 = process_nanY(X_train_preW2Vfirst10, Y_train_preW2Vfirst10)
X_train_preW2Vfirst10 = process_nanX(X_train_preW2Vfirst10)
Y_test_preW2Vfirst10 = process_nanY(X_test_preW2Vfirst10, Y_test_preW2Vfirst10)
X_test_preW2Vfirst10 = process_nanX(X_test_preW2Vfirst10)
```

In [22]:
```
############################## Load the data_4b ##############################
train_data_4b = Train(X_train_preW2Vfirst10, Y_train_preW2Vfirst10-1)
test_data_4b = Test(X_test_preW2Vfirst10, Y_test_preW2Vfirst10-1)

# number of subprocesses to use for data loading
num_workers = 0
# how many samples per batch to load
batch_size = 100
# percentage of training set to use as validation
valid_size = 0.2

# obtain training indices that will be used for validation
num_train = len(train_data_4b)
indices = list(range(num_train))
np.random.shuffle(indices)
split = int(np.floor(valid_size * num_train))
train_idx, valid_idx = indices[split:], indices[:split]

# define samplers for obtaining training and validation batches
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

# prepare data loaders
train_loader_4b = torch.utils.data.DataLoader(train_data_4b, batch_size=batch_size, sampler=train_sampler, nu
valid_loader_4b = torch.utils.data.DataLoader(train_data_4b, batch_size=batch_size, sampler=valid_sampler, nu
test_loader_4b = torch.utils.data.DataLoader(test_data_4b, batch_size=batch_size, num_workers=num_workers)
```

In [23]:
```
############################## Train the model_4b ##############################
start = time.time()

# number of epochs to train the model
n_epochs = 60

# initialize tracker for minimum validation loss
```

```
valid_loss_min = np.Inf # set initial "min" to infinity

for epoch in range(n_epochs):
    # monitor training loss
    train_loss = 0.0
    valid_loss = 0.0

    ###################
    # train the model #
    ###################
    model_4b.train() # prep model for training
    for data, target in train_loader_4b:
        # transfer data and target to GPU
        data, target = data.to(device), target.to(device)
        # clear the gradients of all optimized variables
        optimizer_4b.zero_grad()
        # forward pass: compute predicted outputs by passing inputs to the model
        output = model_4b(data.float())
        # calculate the loss
        loss = criterion(output, target.to(torch.long))
        # backward pass: compute gradient of the loss with respect to model parameters
        loss.backward()
        # perform a single optimization step (parameter update)
        optimizer_4b.step()
        # update running training loss
        train_loss += loss.item()*data.size(0)

    ######################
    # validate the model #
    ######################
    model_4b.eval() # prep model for evaluation
    for data, target in valid_loader_4b:
        # transfer data and target to GPU
        data, target = data.to(device), target.to(device)
        # forward pass: compute predicted outputs by passing inputs to the model
        output = model_4b(data.float())
        # calculate the loss
        loss = criterion(output, target.to(torch.long))
        # update running validation loss
        valid_loss += loss.item()*data.size(0)

    # print training/validation statistics
    # calculate average loss over an epoch
```

```
    train_loss = train_loss/len(train_loader_4b.dataset)
    valid_loss = valid_loss/len(valid_loader_4b.dataset)

    print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
        epoch+1,
        train_loss,
        valid_loss
        ))

    # save model if validation loss has decreased
    if valid_loss <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(
        valid_loss_min,
        valid_loss))
        torch.save(model_4b.state_dict(), 'model.pt')
        valid_loss_min = valid_loss

end = time.time()
print('Time elapsed: %.2f s' % (end - start))
```

```
Epoch: 1        Training Loss: 1.289511         Validation Loss: 0.320628
Validation loss decreased (inf --> 0.320628).  Saving model ...
Epoch: 2        Training Loss: 1.274435         Validation Loss: 0.315348
Validation loss decreased (0.320628 --> 0.315348).  Saving model ...
Epoch: 3        Training Loss: 1.234018         Validation Loss: 0.296562
Validation loss decreased (0.315348 --> 0.296562).  Saving model ...
Epoch: 4        Training Loss: 1.145433         Validation Loss: 0.270334
Validation loss decreased (0.296562 --> 0.270334).  Saving model ...
Epoch: 5        Training Loss: 1.074494         Validation Loss: 0.256175
Validation loss decreased (0.270334 --> 0.256175).  Saving model ...
Epoch: 6        Training Loss: 1.036164         Validation Loss: 0.248015
Validation loss decreased (0.256175 --> 0.248015).  Saving model ...
Epoch: 7        Training Loss: 1.010050         Validation Loss: 0.242429
Validation loss decreased (0.248015 --> 0.242429).  Saving model ...
Epoch: 8        Training Loss: 0.986314         Validation Loss: 0.237203
Validation loss decreased (0.242429 --> 0.237203).  Saving model ...
Epoch: 9        Training Loss: 0.968403         Validation Loss: 0.233014
Validation loss decreased (0.237203 --> 0.233014).  Saving model ...
Epoch: 10       Training Loss: 0.950950         Validation Loss: 0.229083
Validation loss decreased (0.233014 --> 0.229083).  Saving model ...
Epoch: 11       Training Loss: 0.936779         Validation Loss: 0.225619
Validation loss decreased (0.229083 --> 0.225619).  Saving model ...
Epoch: 12       Training Loss: 0.923480         Validation Loss: 0.222807
Validation loss decreased (0.225619 --> 0.222807).  Saving model ...
Epoch: 13       Training Loss: 0.910284         Validation Loss: 0.219944
Validation loss decreased (0.222807 --> 0.219944).  Saving model ...
Epoch: 14       Training Loss: 0.899857         Validation Loss: 0.217853
Validation loss decreased (0.219944 --> 0.217853).  Saving model ...
Epoch: 15       Training Loss: 0.889516         Validation Loss: 0.214809
Validation loss decreased (0.217853 --> 0.214809).  Saving model ...
Epoch: 16       Training Loss: 0.878222         Validation Loss: 0.212580
Validation loss decreased (0.214809 --> 0.212580).  Saving model ...
Epoch: 17       Training Loss: 0.865779         Validation Loss: 0.210392
Validation loss decreased (0.212580 --> 0.210392).  Saving model ...
Epoch: 18       Training Loss: 0.857747         Validation Loss: 0.208768
Validation loss decreased (0.210392 --> 0.208768).  Saving model ...
Epoch: 19       Training Loss: 0.849002         Validation Loss: 0.207060
Validation loss decreased (0.208768 --> 0.207060).  Saving model ...
Epoch: 20       Training Loss: 0.842809         Validation Loss: 0.205862
Validation loss decreased (0.207060 --> 0.205862).  Saving model ...
Epoch: 21       Training Loss: 0.833960         Validation Loss: 0.204378
Validation loss decreased (0.205862 --> 0.204378).  Saving model ...
Epoch: 22       Training Loss: 0.825804         Validation Loss: 0.203159
```

```
Validation loss decreased (0.204378 --> 0.203159).  Saving model ...
Epoch: 23       Training Loss: 0.820146         Validation Loss: 0.202269
Validation loss decreased (0.203159 --> 0.202269).  Saving model ...
Epoch: 24       Training Loss: 0.812477         Validation Loss: 0.201148
Validation loss decreased (0.202269 --> 0.201148).  Saving model ...
Epoch: 25       Training Loss: 0.805287         Validation Loss: 0.199896
Validation loss decreased (0.201148 --> 0.199896).  Saving model ...
Epoch: 26       Training Loss: 0.798398         Validation Loss: 0.199438
Validation loss decreased (0.199896 --> 0.199438).  Saving model ...
Epoch: 27       Training Loss: 0.789745         Validation Loss: 0.198631
Validation loss decreased (0.199438 --> 0.198631).  Saving model ...
Epoch: 28       Training Loss: 0.787119         Validation Loss: 0.198014
Validation loss decreased (0.198631 --> 0.198014).  Saving model ...
Epoch: 29       Training Loss: 0.779115         Validation Loss: 0.197337
Validation loss decreased (0.198014 --> 0.197337).  Saving model ...
Epoch: 30       Training Loss: 0.771816         Validation Loss: 0.197119
Validation loss decreased (0.197337 --> 0.197119).  Saving model ...
Epoch: 31       Training Loss: 0.767442         Validation Loss: 0.196465
Validation loss decreased (0.197119 --> 0.196465).  Saving model ...
Epoch: 32       Training Loss: 0.761873         Validation Loss: 0.195609
Validation loss decreased (0.196465 --> 0.195609).  Saving model ...
Epoch: 33       Training Loss: 0.754038         Validation Loss: 0.195417
Validation loss decreased (0.195609 --> 0.195417).  Saving model ...
Epoch: 34       Training Loss: 0.752689         Validation Loss: 0.195509
Epoch: 35       Training Loss: 0.742244         Validation Loss: 0.194678
Validation loss decreased (0.195417 --> 0.194678).  Saving model ...
Epoch: 36       Training Loss: 0.737640         Validation Loss: 0.194497
Validation loss decreased (0.194678 --> 0.194497).  Saving model ...
Epoch: 37       Training Loss: 0.733939         Validation Loss: 0.194687
Epoch: 38       Training Loss: 0.727856         Validation Loss: 0.194186
Validation loss decreased (0.194497 --> 0.194186).  Saving model ...
Epoch: 39       Training Loss: 0.723471         Validation Loss: 0.194380
Epoch: 40       Training Loss: 0.717033         Validation Loss: 0.194276
Epoch: 41       Training Loss: 0.713816         Validation Loss: 0.194513
Epoch: 42       Training Loss: 0.707955         Validation Loss: 0.194039
Validation loss decreased (0.194186 --> 0.194039).  Saving model ...
Epoch: 43       Training Loss: 0.701363         Validation Loss: 0.194603
Epoch: 44       Training Loss: 0.698734         Validation Loss: 0.194561
Epoch: 45       Training Loss: 0.694149         Validation Loss: 0.194624
Epoch: 46       Training Loss: 0.689464         Validation Loss: 0.194634
Epoch: 47       Training Loss: 0.682761         Validation Loss: 0.195063
Epoch: 48       Training Loss: 0.679266         Validation Loss: 0.195177
Epoch: 49       Training Loss: 0.676229         Validation Loss: 0.195622
```

```
Epoch: 50       Training Loss: 0.666042         Validation Loss: 0.195554
Epoch: 51       Training Loss: 0.665183         Validation Loss: 0.196092
Epoch: 52       Training Loss: 0.659861         Validation Loss: 0.196293
Epoch: 53       Training Loss: 0.653907         Validation Loss: 0.196925
Epoch: 54       Training Loss: 0.649092         Validation Loss: 0.196802
Epoch: 55       Training Loss: 0.646694         Validation Loss: 0.198523
Epoch: 56       Training Loss: 0.641877         Validation Loss: 0.198493
Epoch: 57       Training Loss: 0.637907         Validation Loss: 0.198670
Epoch: 58       Training Loss: 0.634346         Validation Loss: 0.198763
Epoch: 59       Training Loss: 0.629364         Validation Loss: 0.199607
Epoch: 60       Training Loss: 0.623246         Validation Loss: 0.200012
Time elapsed: 131.32 s
```

In [24]:

```
##############################  Calculate & Print the Accurancy_4b on Test#################################
# Load the model with the lowest validation loss
model_4b.load_state_dict(torch.load('model.pt'))
# Calculate the accurancy
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader_4b:
        embeddings, labels = data
        # calculating outputs by running embeddings through the network
        model_4b.to("cpu")
        outputs = model_4b(embeddings.float())
        # the class with the highest score is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(correct/total)
```

```
0.5851
```

- Conclusion
  - FNN model obviously works better than simple models with average vectors as training data.
  - Using the first 10 concatenated vectors as input is better than using the average vectors. The possible reason may be the loss of information when using average vectors. In addition, in most reviews, the first 10 words can correctly determine the classification result of this review.

Recurrent Neural Networks

- Define the Network Architecture (RNN)

In [25]:
```
# define the RNN/GRN architecture
class RNN(nn.Module):
    def __init__(self, model_type = "rnn"):
        super(RNN, self).__init__()

        # define the RNN's parameters
        self.hidden_dim = 20
        self.n_layers = 1
        self.model_type = model_type

        #RNN
        if self.model_type == "gru":
            self.rnn = nn.GRU(300, 20, 1, batch_first=True)
        else:
            self.rnn = nn.RNN(300, 20, 1, batch_first=True, nonlinearity='relu')

        #Outpur layer
        self.fc = nn.Linear(20, 5)

    def forward(self, x):
        # Initialize hidden state with zeros
        h0 = torch.zeros(1, x.size(0), 20).to(device)

        # One time step
        out, hn = self.rnn(x, h0)
        out = self.fc(out[:, -1, :])
        return out

# initialize the NN
model_5a = RNN(model_type = "rnn")
model_5b = RNN(model_type = "gru")
model_5a.cuda()
model_5b.cuda()
print(model_5a)
print(model_5b)

# specify loss function (categorical cross-entropy)
```

```
criterion = nn.CrossEntropyLoss()
# specify optimizer (stochastic gradient descent) and learning rate = 0.01
optimizer_5a = torch.optim.SGD(model_5a.parameters(), lr=0.01)
optimizer_5b = torch.optim.SGD(model_5b.parameters(), lr=0.01)
```

```
RNN(
  (rnn): RNN(300, 20, batch_first=True)
  (fc): Linear(in_features=20, out_features=5, bias=True)
)
RNN(
  (rnn): GRU(300, 20, batch_first=True)
  (fc): Linear(in_features=20, out_features=5, bias=True)
)
```

In [26]:
```
#vStack the first 20 Word2Vec vectors of Google-pretrained Word2Vec Model with trancating & padding
X_train_preW2Vseq20 =np.array(X_train.apply(lambda x: w2v_seq20(google_wv, x)).values.tolist())
X_test_preW2Vseq20 =np.array(X_test.apply(lambda x: w2v_seq20(google_wv, x)).values.tolist())
Y_train_preW2Vseq20 = np.array(Y_train.values.tolist())
Y_test_preW2Vseq20 = np.array(Y_test.values.tolist())
#Delete NaN Vectors with their Labels
Y_train_preW2Vseq20 = process_nanY(X_train_preW2Vseq20, Y_train_preW2Vseq20)
X_train_preW2Vseq20 = process_nanX(X_train_preW2Vseq20)
Y_test_preW2Vseq20 = process_nanY(X_test_preW2Vseq20, Y_test_preW2Vseq20)
X_test_preW2Vseq20 = process_nanX(X_test_preW2Vseq20)
#print(X_train_preW2Vseq20.shape)
```

- (a) Train a simple RNN for sentiment analysis, limiting the review length to 20

In [27]:
```
############################## Load the data_5a ##############################
train_data_5a = Train(X_train_preW2Vseq20, Y_train_preW2Vseq20-1)
test_data_5a = Test(X_test_preW2Vseq20, Y_test_preW2Vseq20-1)

# number of subprocesses to use for data loading
num_workers = 0
# how many samples per batch to load
batch_size = 500
# percentage of training set to use as validation
valid_size = 0.2

# obtain training indices that will be used for validation
num_train = len(train_data_5a)
indices = list(range(num_train))
```

```
np.random.shuffle(indices)
split = int(np.floor(valid_size * num_train))
train_idx, valid_idx = indices[split:], indices[:split]

# define samplers for obtaining training and validation batches
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

# prepare data loaders
train_loader_5a = torch.utils.data.DataLoader(train_data_5a, batch_size=batch_size, sampler=train_sampler, nu
valid_loader_5a = torch.utils.data.DataLoader(train_data_5a, batch_size=batch_size, sampler=valid_sampler, nu
test_loader_5a = torch.utils.data.DataLoader(test_data_5a, batch_size=batch_size, num_workers=num_workers)
```

In [28]:

```
############################## Train the model_5a ##############################
start = time.time()

# number of epochs to train the model
n_epochs = 100

# initialize tracker for minimum validation loss
valid_loss_min = np.Inf # set initial "min" to infinity

for epoch in range(n_epochs):
    # monitor training loss
    train_loss = 0.0
    valid_loss = 0.0

    ###################
    # train the model #
    ###################
    model_5a.train() # prep model for training
    for data, target in train_loader_5a:
        # transfer data and target to GPU
        data, target = data.to(device), target.to(device)
        # clear the gradients of all optimized variables
        optimizer_5a.zero_grad()
        # forward pass: compute predicted outputs by passing inputs to the model
        output = model_5a(data.float())
        # calculate the loss
        loss = criterion(output, target.to(torch.long))
        # backward pass: compute gradient of the loss with respect to model parameters
        loss.backward()
        # perform a single optimization step (parameter update)
```

```
        optimizer_5a.step()
        # update running training loss
        train_loss += loss.item()*data.size(0)

    ######################
    # validate the model #
    ######################
    model_5a.eval() # prep model for evaluation
    for data, target in valid_loader_5a:
        # transfer data and target to GPU
        data, target = data.to(device), target.to(device)
        # forward pass: compute predicted outputs by passing inputs to the model
        output = model_5a(data.float())
        # calculate the loss
        loss = criterion(output, target.to(torch.long))
        # update running validation loss
        valid_loss += loss.item()*data.size(0)

    # print training/validation statistics
    # calculate average loss over an epoch
    train_loss = train_loss/len(train_loader_5a.dataset)
    valid_loss = valid_loss/len(valid_loader_5a.dataset)

    print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
        epoch+1,
        train_loss,
        valid_loss
        ))

    # save model if validation loss has decreased
    if valid_loss <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(
        valid_loss_min,
        valid_loss))
        torch.save(model_5a.state_dict(), 'model.pt')
        valid_loss_min = valid_loss

end = time.time()
print('Time elapsed: %.2f s' % (end - start))
```

```
Epoch: 1        Training Loss: 1.292324         Validation Loss: 0.322846
Validation loss decreased (inf --> 0.322846).  Saving model ...
Epoch: 2        Training Loss: 1.290458         Validation Loss: 0.322543
Validation loss decreased (0.322846 --> 0.322543).  Saving model ...
Epoch: 3        Training Loss: 1.289515         Validation Loss: 0.322386
Validation loss decreased (0.322543 --> 0.322386).  Saving model ...
Epoch: 4        Training Loss: 1.288990         Validation Loss: 0.322294
Validation loss decreased (0.322386 --> 0.322294).  Saving model ...
Epoch: 5        Training Loss: 1.288644         Validation Loss: 0.322227
Validation loss decreased (0.322294 --> 0.322227).  Saving model ...
Epoch: 6        Training Loss: 1.288379         Validation Loss: 0.322172
Validation loss decreased (0.322227 --> 0.322172).  Saving model ...
Epoch: 7        Training Loss: 1.288153         Validation Loss: 0.322122
Validation loss decreased (0.322172 --> 0.322122).  Saving model ...
Epoch: 8        Training Loss: 1.287949         Validation Loss: 0.322075
Validation loss decreased (0.322122 --> 0.322075).  Saving model ...
Epoch: 9        Training Loss: 1.287757         Validation Loss: 0.322032
Validation loss decreased (0.322075 --> 0.322032).  Saving model ...
Epoch: 10       Training Loss: 1.287571         Validation Loss: 0.321989
Validation loss decreased (0.322032 --> 0.321989).  Saving model ...
Epoch: 11       Training Loss: 1.287394         Validation Loss: 0.321948
Validation loss decreased (0.321989 --> 0.321948).  Saving model ...
Epoch: 12       Training Loss: 1.287220         Validation Loss: 0.321907
Validation loss decreased (0.321948 --> 0.321907).  Saving model ...
Epoch: 13       Training Loss: 1.287047         Validation Loss: 0.321868
Validation loss decreased (0.321907 --> 0.321868).  Saving model ...
Epoch: 14       Training Loss: 1.286879         Validation Loss: 0.321827
Validation loss decreased (0.321868 --> 0.321827).  Saving model ...
Epoch: 15       Training Loss: 1.286709         Validation Loss: 0.321788
Validation loss decreased (0.321827 --> 0.321788).  Saving model ...
Epoch: 16       Training Loss: 1.286538         Validation Loss: 0.321747
Validation loss decreased (0.321788 --> 0.321747).  Saving model ...
Epoch: 17       Training Loss: 1.286371         Validation Loss: 0.321707
Validation loss decreased (0.321747 --> 0.321707).  Saving model ...
Epoch: 18       Training Loss: 1.286202         Validation Loss: 0.321666
Validation loss decreased (0.321707 --> 0.321666).  Saving model ...
Epoch: 19       Training Loss: 1.286034         Validation Loss: 0.321625
Validation loss decreased (0.321666 --> 0.321625).  Saving model ...
Epoch: 20       Training Loss: 1.285867         Validation Loss: 0.321583
Validation loss decreased (0.321625 --> 0.321583).  Saving model ...
Epoch: 21       Training Loss: 1.285694         Validation Loss: 0.321541
Validation loss decreased (0.321583 --> 0.321541).  Saving model ...
Epoch: 22       Training Loss: 1.285519         Validation Loss: 0.321499
```

```
Validation loss decreased (0.321541 --> 0.321499).  Saving model ...
Epoch: 23       Training Loss: 1.285342         Validation Loss: 0.321454
Validation loss decreased (0.321499 --> 0.321454).  Saving model ...
Epoch: 24       Training Loss: 1.285158         Validation Loss: 0.321409
Validation loss decreased (0.321454 --> 0.321409).  Saving model ...
Epoch: 25       Training Loss: 1.284972         Validation Loss: 0.321361
Validation loss decreased (0.321409 --> 0.321361).  Saving model ...
Epoch: 26       Training Loss: 1.284779         Validation Loss: 0.321314
Validation loss decreased (0.321361 --> 0.321314).  Saving model ...
Epoch: 27       Training Loss: 1.284585         Validation Loss: 0.321264
Validation loss decreased (0.321314 --> 0.321264).  Saving model ...
Epoch: 28       Training Loss: 1.284384         Validation Loss: 0.321213
Validation loss decreased (0.321264 --> 0.321213).  Saving model ...
Epoch: 29       Training Loss: 1.284175         Validation Loss: 0.321160
Validation loss decreased (0.321213 --> 0.321160).  Saving model ...
Epoch: 30       Training Loss: 1.283958         Validation Loss: 0.321107
Validation loss decreased (0.321160 --> 0.321107).  Saving model ...
Epoch: 31       Training Loss: 1.283736         Validation Loss: 0.321051
Validation loss decreased (0.321107 --> 0.321051).  Saving model ...
Epoch: 32       Training Loss: 1.283504         Validation Loss: 0.320992
Validation loss decreased (0.321051 --> 0.320992).  Saving model ...
Epoch: 33       Training Loss: 1.283261         Validation Loss: 0.320929
Validation loss decreased (0.320992 --> 0.320929).  Saving model ...
Epoch: 34       Training Loss: 1.283010         Validation Loss: 0.320865
Validation loss decreased (0.320929 --> 0.320865).  Saving model ...
Epoch: 35       Training Loss: 1.282743         Validation Loss: 0.320796
Validation loss decreased (0.320865 --> 0.320796).  Saving model ...
Epoch: 36       Training Loss: 1.282459         Validation Loss: 0.320724
Validation loss decreased (0.320796 --> 0.320724).  Saving model ...
Epoch: 37       Training Loss: 1.282162         Validation Loss: 0.320647
Validation loss decreased (0.320724 --> 0.320647).  Saving model ...
Epoch: 38       Training Loss: 1.281842         Validation Loss: 0.320563
Validation loss decreased (0.320647 --> 0.320563).  Saving model ...
Epoch: 39       Training Loss: 1.281496         Validation Loss: 0.320474
Validation loss decreased (0.320563 --> 0.320474).  Saving model ...
Epoch: 40       Training Loss: 1.281120         Validation Loss: 0.320373
Validation loss decreased (0.320474 --> 0.320373).  Saving model ...
Epoch: 41       Training Loss: 1.280715         Validation Loss: 0.320266
Validation loss decreased (0.320373 --> 0.320266).  Saving model ...
Epoch: 42       Training Loss: 1.280263         Validation Loss: 0.320144
Validation loss decreased (0.320266 --> 0.320144).  Saving model ...
Epoch: 43       Training Loss: 1.279756         Validation Loss: 0.320010
Validation loss decreased (0.320144 --> 0.320010).  Saving model ...
```

```
Epoch: 44       Training Loss: 1.279174         Validation Loss: 0.319851
Validation loss decreased (0.320010 --> 0.319851).  Saving model ...
Epoch: 45       Training Loss: 1.278488         Validation Loss: 0.319660
Validation loss decreased (0.319851 --> 0.319660).  Saving model ...
Epoch: 46       Training Loss: 1.277632         Validation Loss: 0.319414
Validation loss decreased (0.319660 --> 0.319414).  Saving model ...
Epoch: 47       Training Loss: 1.276475         Validation Loss: 0.319065
Validation loss decreased (0.319414 --> 0.319065).  Saving model ...
Epoch: 48       Training Loss: 1.274612         Validation Loss: 0.318447
Validation loss decreased (0.319065 --> 0.318447).  Saving model ...
Epoch: 49       Training Loss: 1.269972         Validation Loss: 0.316061
Validation loss decreased (0.318447 --> 0.316061).  Saving model ...
Epoch: 50       Training Loss: 1.240116         Validation Loss: 0.303339
Validation loss decreased (0.316061 --> 0.303339).  Saving model ...
Epoch: 51       Training Loss: 1.189503         Validation Loss: 0.294290
Validation loss decreased (0.303339 --> 0.294290).  Saving model ...
Epoch: 52       Training Loss: 1.158208         Validation Loss: 0.280646
Validation loss decreased (0.294290 --> 0.280646).  Saving model ...
Epoch: 53       Training Loss: 1.131312         Validation Loss: 0.287574
Epoch: 54       Training Loss: 1.105968         Validation Loss: 0.273260
Validation loss decreased (0.280646 --> 0.273260).  Saving model ...
Epoch: 55       Training Loss: 1.090948         Validation Loss: 0.264989
Validation loss decreased (0.273260 --> 0.264989).  Saving model ...
Epoch: 56       Training Loss: 1.079139         Validation Loss: 0.271201
Epoch: 57       Training Loss: 1.068071         Validation Loss: 0.259775
Validation loss decreased (0.264989 --> 0.259775).  Saving model ...
Epoch: 58       Training Loss: 1.055752         Validation Loss: 0.266826
Epoch: 59       Training Loss: 1.043750         Validation Loss: 0.256900
Validation loss decreased (0.259775 --> 0.256900).  Saving model ...
Epoch: 60       Training Loss: 1.034002         Validation Loss: 0.251311
Validation loss decreased (0.256900 --> 0.251311).  Saving model ...
Epoch: 61       Training Loss: 1.030592         Validation Loss: 0.260324
Epoch: 62       Training Loss: 1.019257         Validation Loss: 0.259728
Epoch: 63       Training Loss: 1.015429         Validation Loss: 0.253378
Epoch: 64       Training Loss: 1.009354         Validation Loss: 0.276417
Epoch: 65       Training Loss: 1.002478         Validation Loss: 0.247540
Validation loss decreased (0.251311 --> 0.247540).  Saving model ...
Epoch: 66       Training Loss: 0.997006         Validation Loss: 0.245700
Validation loss decreased (0.247540 --> 0.245700).  Saving model ...
Epoch: 67       Training Loss: 0.991932         Validation Loss: 0.253601
Epoch: 68       Training Loss: 0.984552         Validation Loss: 0.251511
Epoch: 69       Training Loss: 0.982645         Validation Loss: 0.243213
Validation loss decreased (0.245700 --> 0.243213).  Saving model ...
```

```
Epoch: 70       Training Loss: 0.974676         Validation Loss: 0.259262
Epoch: 71       Training Loss: 0.971620         Validation Loss: 0.237583
Validation loss decreased (0.243213 --> 0.237583).  Saving model ...
Epoch: 72       Training Loss: 0.971615         Validation Loss: 0.245837
Epoch: 73       Training Loss: 0.967250         Validation Loss: 0.244364
Epoch: 74       Training Loss: 0.963155         Validation Loss: 0.239088
Epoch: 75       Training Loss: 0.959306         Validation Loss: 0.246148
Epoch: 76       Training Loss: 0.954288         Validation Loss: 0.233712
Validation loss decreased (0.237583 --> 0.233712).  Saving model ...
Epoch: 77       Training Loss: 0.951398         Validation Loss: 0.234757
Epoch: 78       Training Loss: 0.945942         Validation Loss: 0.231624
Validation loss decreased (0.233712 --> 0.231624).  Saving model ...
Epoch: 79       Training Loss: 0.945500         Validation Loss: 0.231455
Validation loss decreased (0.231624 --> 0.231455).  Saving model ...
Epoch: 80       Training Loss: 0.943542         Validation Loss: 0.233064
Epoch: 81       Training Loss: 0.936307         Validation Loss: 0.233948
Epoch: 82       Training Loss: 0.939050         Validation Loss: 0.229369
Validation loss decreased (0.231455 --> 0.229369).  Saving model ...
Epoch: 83       Training Loss: 0.934525         Validation Loss: 0.232081
Epoch: 84       Training Loss: 0.929741         Validation Loss: 0.227969
Validation loss decreased (0.229369 --> 0.227969).  Saving model ...
Epoch: 85       Training Loss: 0.932190         Validation Loss: 0.227221
Validation loss decreased (0.227969 --> 0.227221).  Saving model ...
Epoch: 86       Training Loss: 0.926006         Validation Loss: 0.231358
Epoch: 87       Training Loss: 0.923973         Validation Loss: 0.227742
Epoch: 88       Training Loss: 0.921580         Validation Loss: 0.229076
Epoch: 89       Training Loss: 0.923507         Validation Loss: 0.236910
Epoch: 90       Training Loss: 0.919878         Validation Loss: 0.225733
Validation loss decreased (0.227221 --> 0.225733).  Saving model ...
Epoch: 91       Training Loss: 0.918799         Validation Loss: 0.232710
Epoch: 92       Training Loss: 0.917578         Validation Loss: 0.223989
Validation loss decreased (0.225733 --> 0.223989).  Saving model ...
Epoch: 93       Training Loss: 0.909126         Validation Loss: 0.231377
Epoch: 94       Training Loss: 0.908027         Validation Loss: 0.254803
Epoch: 95       Training Loss: 0.905279         Validation Loss: 0.224945
Epoch: 96       Training Loss: 0.907633         Validation Loss: 0.232085
Epoch: 97       Training Loss: 0.904948         Validation Loss: 0.236172
Epoch: 98       Training Loss: 0.904227         Validation Loss: 0.220569
Validation loss decreased (0.223989 --> 0.220569).  Saving model ...
Epoch: 99       Training Loss: 0.901072         Validation Loss: 0.251004
Epoch: 100      Training Loss: 0.899603         Validation Loss: 0.231154
Time elapsed: 247.01 s
```

In [29]:
```
############################  Calculate & Print the Accurancy_5a on Test#################################
# Load the model with the lowest validation loss
model_5a.load_state_dict(torch.load('model.pt'))
# Calculate the accurancy
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader_5a:
        embeddings, labels = data
        # transfer data and target to GPU
        embeddings, labels = embeddings.to(device), labels.to(device)
        # calculating outputs by running embeddings through the network
        model_5a.to(device)
        outputs = model_5a(embeddings.float())
        # the class with the highest score is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(correct/total)
```

```
0.5171
```

- (b) Repeat part (a) by considering a gated recurrent unit cell

In [30]:
```
############################ Load the data_5b ##############################
train_data_5b = Train(X_train_preW2Vseq20, Y_train_preW2Vseq20-1)
test_data_5b = Test(X_test_preW2Vseq20, Y_test_preW2Vseq20-1)

# number of subprocesses to use for data loading
num_workers = 0
# how many samples per batch to load
batch_size = 500
# percentage of training set to use as validation
valid_size = 0.2

# obtain training indices that will be used for validation
num_train = len(train_data_5b)
indices = list(range(num_train))
np.random.shuffle(indices)
split = int(np.floor(valid_size * num_train))
train_idx, valid_idx = indices[split:], indices[:split]
```

```
# define samplers for obtaining training and validation batches
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

# prepare data loaders
train_loader_5b = torch.utils.data.DataLoader(train_data_5b, batch_size=batch_size, sampler=train_sampler, nu
valid_loader_5b = torch.utils.data.DataLoader(train_data_5b, batch_size=batch_size, sampler=valid_sampler, nu
test_loader_5b = torch.utils.data.DataLoader(test_data_5b, batch_size=batch_size, num_workers=num_workers)
```

In [31]:
```
############################## Train the model_5b ##############################
start = time.time()

# number of epochs to train the model
n_epochs = 100

# initialize tracker for minimum validation loss
valid_loss_min = np.Inf # set initial "min" to infinity

for epoch in range(n_epochs):
    # monitor training loss
    train_loss = 0.0
    valid_loss = 0.0

    ###################
    # train the model #
    ###################
    model_5b.train() # prep model for training
    for data, target in train_loader_5b:
        # transfer data and target to GPU
        data, target = data.to(device), target.to(device)
        # clear the gradients of all optimized variables
        optimizer_5b.zero_grad()
        # forward pass: compute predicted outputs by passing inputs to the model
        output = model_5b(data.float())
        # calculate the loss
        loss = criterion(output, target.to(torch.long))
        # backward pass: compute gradient of the loss with respect to model parameters
        loss.backward()
        # perform a single optimization step (parameter update)
        optimizer_5b.step()
        # update running training loss
        train_loss += loss.item()*data.size(0)
```

```
    ######################
    # validate the model #
    ######################
    model_5b.eval() # prep model for evaluation
    for data, target in valid_loader_5b:
        # transfer data and target to GPU
        data, target = data.to(device), target.to(device)
        # forward pass: compute predicted outputs by passing inputs to the model
        output = model_5b(data.float())
        # calculate the loss
        loss = criterion(output, target.to(torch.long))
        # update running validation loss
        valid_loss += loss.item()*data.size(0)

    # print training/validation statistics
    # calculate average loss over an epoch
    train_loss = train_loss/len(train_loader_5b.dataset)
    valid_loss = valid_loss/len(valid_loader_5b.dataset)

    print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
        epoch+1,
        train_loss,
        valid_loss
        ))

    # save model if validation loss has decreased
    if valid_loss <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(
        valid_loss_min,
        valid_loss))
        torch.save(model_5b.state_dict(), 'model.pt')
        valid_loss_min = valid_loss

end = time.time()
print('Time elapsed: %.2f s' % (end - start))
```

```
Epoch: 1        Training Loss: 1.297767         Validation Loss: 0.323643
Validation loss decreased (inf --> 0.323643).  Saving model ...
Epoch: 2        Training Loss: 1.292575         Validation Loss: 0.322852
Validation loss decreased (0.323643 --> 0.322852).  Saving model ...
Epoch: 3        Training Loss: 1.290530         Validation Loss: 0.322515
Validation loss decreased (0.322852 --> 0.322515).  Saving model ...
Epoch: 4        Training Loss: 1.289610         Validation Loss: 0.322346
Validation loss decreased (0.322515 --> 0.322346).  Saving model ...
Epoch: 5        Training Loss: 1.289103         Validation Loss: 0.322238
Validation loss decreased (0.322346 --> 0.322238).  Saving model ...
Epoch: 6        Training Loss: 1.288743         Validation Loss: 0.322155
Validation loss decreased (0.322238 --> 0.322155).  Saving model ...
Epoch: 7        Training Loss: 1.288443         Validation Loss: 0.322082
Validation loss decreased (0.322155 --> 0.322082).  Saving model ...
Epoch: 8        Training Loss: 1.288166         Validation Loss: 0.322015
Validation loss decreased (0.322082 --> 0.322015).  Saving model ...
Epoch: 9        Training Loss: 1.287900         Validation Loss: 0.321950
Validation loss decreased (0.322015 --> 0.321950).  Saving model ...
Epoch: 10       Training Loss: 1.287641         Validation Loss: 0.321886
Validation loss decreased (0.321950 --> 0.321886).  Saving model ...
Epoch: 11       Training Loss: 1.287386         Validation Loss: 0.321823
Validation loss decreased (0.321886 --> 0.321823).  Saving model ...
Epoch: 12       Training Loss: 1.287136         Validation Loss: 0.321762
Validation loss decreased (0.321823 --> 0.321762).  Saving model ...
Epoch: 13       Training Loss: 1.286889         Validation Loss: 0.321701
Validation loss decreased (0.321762 --> 0.321701).  Saving model ...
Epoch: 14       Training Loss: 1.286645         Validation Loss: 0.321640
Validation loss decreased (0.321701 --> 0.321640).  Saving model ...
Epoch: 15       Training Loss: 1.286404         Validation Loss: 0.321581
Validation loss decreased (0.321640 --> 0.321581).  Saving model ...
Epoch: 16       Training Loss: 1.286163         Validation Loss: 0.321522
Validation loss decreased (0.321581 --> 0.321522).  Saving model ...
Epoch: 17       Training Loss: 1.285925         Validation Loss: 0.321463
Validation loss decreased (0.321522 --> 0.321463).  Saving model ...
Epoch: 18       Training Loss: 1.285687         Validation Loss: 0.321405
Validation loss decreased (0.321463 --> 0.321405).  Saving model ...
Epoch: 19       Training Loss: 1.285451         Validation Loss: 0.321347
Validation loss decreased (0.321405 --> 0.321347).  Saving model ...
Epoch: 20       Training Loss: 1.285214         Validation Loss: 0.321289
Validation loss decreased (0.321347 --> 0.321289).  Saving model ...
Epoch: 21       Training Loss: 1.284979         Validation Loss: 0.321231
Validation loss decreased (0.321289 --> 0.321231).  Saving model ...
Epoch: 22       Training Loss: 1.284742         Validation Loss: 0.321174
```

```
Validation loss decreased (0.321231 --> 0.321174).  Saving model ...
Epoch: 23       Training Loss: 1.284503         Validation Loss: 0.321115
Validation loss decreased (0.321174 --> 0.321115).  Saving model ...
Epoch: 24       Training Loss: 1.284265         Validation Loss: 0.321056
Validation loss decreased (0.321115 --> 0.321056).  Saving model ...
Epoch: 25       Training Loss: 1.284026         Validation Loss: 0.320998
Validation loss decreased (0.321056 --> 0.320998).  Saving model ...
Epoch: 26       Training Loss: 1.283784         Validation Loss: 0.320937
Validation loss decreased (0.320998 --> 0.320937).  Saving model ...
Epoch: 27       Training Loss: 1.283543         Validation Loss: 0.320878
Validation loss decreased (0.320937 --> 0.320878).  Saving model ...
Epoch: 28       Training Loss: 1.283300         Validation Loss: 0.320818
Validation loss decreased (0.320878 --> 0.320818).  Saving model ...
Epoch: 29       Training Loss: 1.283052         Validation Loss: 0.320757
Validation loss decreased (0.320818 --> 0.320757).  Saving model ...
Epoch: 30       Training Loss: 1.282800         Validation Loss: 0.320695
Validation loss decreased (0.320757 --> 0.320695).  Saving model ...
Epoch: 31       Training Loss: 1.282548         Validation Loss: 0.320633
Validation loss decreased (0.320695 --> 0.320633).  Saving model ...
Epoch: 32       Training Loss: 1.282291         Validation Loss: 0.320569
Validation loss decreased (0.320633 --> 0.320569).  Saving model ...
Epoch: 33       Training Loss: 1.282033         Validation Loss: 0.320505
Validation loss decreased (0.320569 --> 0.320505).  Saving model ...
Epoch: 34       Training Loss: 1.281769         Validation Loss: 0.320441
Validation loss decreased (0.320505 --> 0.320441).  Saving model ...
Epoch: 35       Training Loss: 1.281500         Validation Loss: 0.320375
Validation loss decreased (0.320441 --> 0.320375).  Saving model ...
Epoch: 36       Training Loss: 1.281229         Validation Loss: 0.320305
Validation loss decreased (0.320375 --> 0.320305).  Saving model ...
Epoch: 37       Training Loss: 1.280953         Validation Loss: 0.320237
Validation loss decreased (0.320305 --> 0.320237).  Saving model ...
Epoch: 38       Training Loss: 1.280669         Validation Loss: 0.320167
Validation loss decreased (0.320237 --> 0.320167).  Saving model ...
Epoch: 39       Training Loss: 1.280384         Validation Loss: 0.320096
Validation loss decreased (0.320167 --> 0.320096).  Saving model ...
Epoch: 40       Training Loss: 1.280093         Validation Loss: 0.320024
Validation loss decreased (0.320096 --> 0.320024).  Saving model ...
Epoch: 41       Training Loss: 1.279795         Validation Loss: 0.319950
Validation loss decreased (0.320024 --> 0.319950).  Saving model ...
Epoch: 42       Training Loss: 1.279491         Validation Loss: 0.319874
Validation loss decreased (0.319950 --> 0.319874).  Saving model ...
Epoch: 43       Training Loss: 1.279180         Validation Loss: 0.319796
Validation loss decreased (0.319874 --> 0.319796).  Saving model ...
```

```
Epoch: 44       Training Loss: 1.278862         Validation Loss: 0.319716
Validation loss decreased (0.319796 --> 0.319716).  Saving model ...
Epoch: 45       Training Loss: 1.278537         Validation Loss: 0.319633
Validation loss decreased (0.319716 --> 0.319633).  Saving model ...
Epoch: 46       Training Loss: 1.278205         Validation Loss: 0.319550
Validation loss decreased (0.319633 --> 0.319550).  Saving model ...
Epoch: 47       Training Loss: 1.277864         Validation Loss: 0.319465
Validation loss decreased (0.319550 --> 0.319465).  Saving model ...
Epoch: 48       Training Loss: 1.277513         Validation Loss: 0.319378
Validation loss decreased (0.319465 --> 0.319378).  Saving model ...
Epoch: 49       Training Loss: 1.277155         Validation Loss: 0.319288
Validation loss decreased (0.319378 --> 0.319288).  Saving model ...
Epoch: 50       Training Loss: 1.276788         Validation Loss: 0.319195
Validation loss decreased (0.319288 --> 0.319195).  Saving model ...
Epoch: 51       Training Loss: 1.276410         Validation Loss: 0.319100
Validation loss decreased (0.319195 --> 0.319100).  Saving model ...
Epoch: 52       Training Loss: 1.276017         Validation Loss: 0.319004
Validation loss decreased (0.319100 --> 0.319004).  Saving model ...
Epoch: 53       Training Loss: 1.275620         Validation Loss: 0.318903
Validation loss decreased (0.319004 --> 0.318903).  Saving model ...
Epoch: 54       Training Loss: 1.275206         Validation Loss: 0.318797
Validation loss decreased (0.318903 --> 0.318797).  Saving model ...
Epoch: 55       Training Loss: 1.274782         Validation Loss: 0.318689
Validation loss decreased (0.318797 --> 0.318689).  Saving model ...
Epoch: 56       Training Loss: 1.274345         Validation Loss: 0.318578
Validation loss decreased (0.318689 --> 0.318578).  Saving model ...
Epoch: 57       Training Loss: 1.273890         Validation Loss: 0.318462
Validation loss decreased (0.318578 --> 0.318462).  Saving model ...
Epoch: 58       Training Loss: 1.273422         Validation Loss: 0.318342
Validation loss decreased (0.318462 --> 0.318342).  Saving model ...
Epoch: 59       Training Loss: 1.272934         Validation Loss: 0.318217
Validation loss decreased (0.318342 --> 0.318217).  Saving model ...
Epoch: 60       Training Loss: 1.272431         Validation Loss: 0.318089
Validation loss decreased (0.318217 --> 0.318089).  Saving model ...
Epoch: 61       Training Loss: 1.271908         Validation Loss: 0.317955
Validation loss decreased (0.318089 --> 0.317955).  Saving model ...
Epoch: 62       Training Loss: 1.271359         Validation Loss: 0.317812
Validation loss decreased (0.317955 --> 0.317812).  Saving model ...
Epoch: 63       Training Loss: 1.270788         Validation Loss: 0.317665
Validation loss decreased (0.317812 --> 0.317665).  Saving model ...
Epoch: 64       Training Loss: 1.270194         Validation Loss: 0.317512
Validation loss decreased (0.317665 --> 0.317512).  Saving model ...
Epoch: 65       Training Loss: 1.269567         Validation Loss: 0.317351
```

```
Validation loss decreased (0.317512 --> 0.317351).  Saving model ...
Epoch: 66       Training Loss: 1.268908         Validation Loss: 0.317181
Validation loss decreased (0.317351 --> 0.317181).  Saving model ...
Epoch: 67       Training Loss: 1.268212         Validation Loss: 0.317002
Validation loss decreased (0.317181 --> 0.317002).  Saving model ...
Epoch: 68       Training Loss: 1.267476         Validation Loss: 0.316808
Validation loss decreased (0.317002 --> 0.316808).  Saving model ...
Epoch: 69       Training Loss: 1.266692         Validation Loss: 0.316601
Validation loss decreased (0.316808 --> 0.316601).  Saving model ...
Epoch: 70       Training Loss: 1.265851         Validation Loss: 0.316382
Validation loss decreased (0.316601 --> 0.316382).  Saving model ...
Epoch: 71       Training Loss: 1.264945         Validation Loss: 0.316140
Validation loss decreased (0.316382 --> 0.316140).  Saving model ...
Epoch: 72       Training Loss: 1.263962         Validation Loss: 0.315879
Validation loss decreased (0.316140 --> 0.315879).  Saving model ...
Epoch: 73       Training Loss: 1.262887         Validation Loss: 0.315592
Validation loss decreased (0.315879 --> 0.315592).  Saving model ...
Epoch: 74       Training Loss: 1.261696         Validation Loss: 0.315279
Validation loss decreased (0.315592 --> 0.315279).  Saving model ...
Epoch: 75       Training Loss: 1.260366         Validation Loss: 0.314914
Validation loss decreased (0.315279 --> 0.314914).  Saving model ...
Epoch: 76       Training Loss: 1.258842         Validation Loss: 0.314495
Validation loss decreased (0.314914 --> 0.314495).  Saving model ...
Epoch: 77       Training Loss: 1.257090         Validation Loss: 0.314011
Validation loss decreased (0.314495 --> 0.314011).  Saving model ...
Epoch: 78       Training Loss: 1.255000         Validation Loss: 0.313431
Validation loss decreased (0.314011 --> 0.313431).  Saving model ...
Epoch: 79       Training Loss: 1.252431         Validation Loss: 0.312698
Validation loss decreased (0.313431 --> 0.312698).  Saving model ...
Epoch: 80       Training Loss: 1.249130         Validation Loss: 0.311736
Validation loss decreased (0.312698 --> 0.311736).  Saving model ...
Epoch: 81       Training Loss: 1.244634         Validation Loss: 0.310379
Validation loss decreased (0.311736 --> 0.310379).  Saving model ...
Epoch: 82       Training Loss: 1.237957         Validation Loss: 0.308279
Validation loss decreased (0.310379 --> 0.308279).  Saving model ...
Epoch: 83       Training Loss: 1.226617         Validation Loss: 0.304440
Validation loss decreased (0.308279 --> 0.304440).  Saving model ...
Epoch: 84       Training Loss: 1.203611         Validation Loss: 0.296006
Validation loss decreased (0.304440 --> 0.296006).  Saving model ...
Epoch: 85       Training Loss: 1.155616         Validation Loss: 0.281637
Validation loss decreased (0.296006 --> 0.281637).  Saving model ...
Epoch: 86       Training Loss: 1.107987         Validation Loss: 0.273769
Validation loss decreased (0.281637 --> 0.273769).  Saving model ...
```

```
Epoch: 87       Training Loss: 1.087072         Validation Loss: 0.269973
Validation loss decreased (0.273769 --> 0.269973).  Saving model ...
Epoch: 88       Training Loss: 1.074214         Validation Loss: 0.267203
Validation loss decreased (0.269973 --> 0.267203).  Saving model ...
Epoch: 89       Training Loss: 1.064148         Validation Loss: 0.264845
Validation loss decreased (0.267203 --> 0.264845).  Saving model ...
Epoch: 90       Training Loss: 1.055735         Validation Loss: 0.262873
Validation loss decreased (0.264845 --> 0.262873).  Saving model ...
Epoch: 91       Training Loss: 1.048016         Validation Loss: 0.261062
Validation loss decreased (0.262873 --> 0.261062).  Saving model ...
Epoch: 92       Training Loss: 1.041087         Validation Loss: 0.259545
Validation loss decreased (0.261062 --> 0.259545).  Saving model ...
Epoch: 93       Training Loss: 1.034706         Validation Loss: 0.257947
Validation loss decreased (0.259545 --> 0.257947).  Saving model ...
Epoch: 94       Training Loss: 1.028736         Validation Loss: 0.256612
Validation loss decreased (0.257947 --> 0.256612).  Saving model ...
Epoch: 95       Training Loss: 1.023008         Validation Loss: 0.255250
Validation loss decreased (0.256612 --> 0.255250).  Saving model ...
Epoch: 96       Training Loss: 1.017530         Validation Loss: 0.254148
Validation loss decreased (0.255250 --> 0.254148).  Saving model ...
Epoch: 97       Training Loss: 1.012366         Validation Loss: 0.253067
Validation loss decreased (0.254148 --> 0.253067).  Saving model ...
Epoch: 98       Training Loss: 1.007167         Validation Loss: 0.252144
Validation loss decreased (0.253067 --> 0.252144).  Saving model ...
Epoch: 99       Training Loss: 1.002186         Validation Loss: 0.250422
Validation loss decreased (0.252144 --> 0.250422).  Saving model ...
Epoch: 100      Training Loss: 0.997231         Validation Loss: 0.249797
Validation loss decreased (0.250422 --> 0.249797).  Saving model ...
Time elapsed: 249.07 s
```

In [32]:

```
##############################  Calculate & Print the Accurancy_5b on Test#################################
# Load the model with the lowest validation loss
model_5b.load_state_dict(torch.load('model.pt'))
# Calculate the accurancy
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader_5b:
        embeddings, labels = data
        # transfer data and target to GPU
        embeddings, labels = embeddings.to(device), labels.to(device)
        # calculating outputs by running embeddings through the network
        model_5b.to(device)
```

```
        outputs = model_5b(embeddings.float())
        # the class with the highest score is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(correct/total)
```

```
0.4332
```

- Conclusion
  - RNN model(0.52) works worse than FNN models(0.57 / 0.59) here.
  - GRU model(0.43) (Gated Recurrent Neural Network) even work worse than simple RNN model(0.52).
  - The poor results of RNN may also be related to the network settings of RNN.

Accuracy values for 6 cases:

1. Perceptron -> 0.46
2. SVM -> 0.56
3. FNN(average Word2Vec vectors) -> 0.57
4. FNN(first 10 Word2Vec vectors) -> 0.59
5. RNN -> 0.52
6. GRN -> 0.43