```
#useful imports

import numpy as np
from numpy.random import randint,rand,seed,normal,permutation
import torch
import torchvision
import torch.nn.functional as F
from torch.utils.data import random_split,Dataset,DataLoader
from torchvision import datasets, transforms
from torch import nn, optim

from torchvision.datasets import MNIST
import torchvision.transforms as T


from keras.datasets import mnist
import matplotlib.pyplot as plt
from copy import deepcopy
from tqdm import tqdm

from scipy.special import softmax
import spacy
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

```
# data_dir = 'https://drive.google.com/drive/folders/1stS7Uj-GAtJASlgUFxyz0JM58dnoTA8L?usp=drive_link'
```

```
# from gensim.scripts.glove2word2vec import glove2word2vec
# glove_input_file = 'glove.txt'
# word2vec_output_file = 'word2vec.txt'
# glove2word2vec(glove_input_file, word2vec_output_file)
```

```
# from gensim.scripts.glove2word2vec import glove2word2vec
# glove_input_file = 'https://drive.google.com/file/d/18aLgOmWo7bFidz3cJES52m51CNGS0SoQ/view?usp=sharing'
# word2vec_output_file = 'glove.6B.100d.txt.word2vec'
# glove2word2vec(glove_input_file, word2vec_output_file)
```

```
from gensim.scripts.glove2word2vec import glove2word2vec
glove_input_file ='/content/drive/My Drive/test/glove.6B.100d.txt'
word2vec_output_file = 'glove.6B.100d.txt.word2vec'
glove2word2vec(glove_input_file, word2vec_output_file)
```

```
    <ipython-input-6-9e4f2c338670>:4: DeprecationWarning: Call to deprecated `glove2word2vec` (KeyedVectors.load_word2vec_fo
      glove2word2vec(glove_input_file, word2vec_output_file)
    (400000, 100)
```

```
from gensim.models import KeyedVectors
# load the Stanford GloVe model
filename = 'glove.6B.100d.txt.word2vec'
model = KeyedVectors.load_word2vec_format(filename, binary=False)
# calculate: (king - man) + woman = ?
result = model.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
print(result)
```

```
    [('queen', 0.7698540687561035)]
```

```
emails_raw = pd.read_csv('/content/drive/My Drive/test/enron_spam_ham.csv').to_numpy()
```

```
# file_path = '/content/drive/My Drive/test/message.txt'

# # Open and read the file
# with open(file_path, 'r') as file:
#     content = file.read()

# # Print the file content
# print(content)
```

```
emails_raw[0]
```

```
array(["Subject: sevil yamin  anne ,  vasant sent this information to norma . i shall fwd his message to you .  vince
- - - - - - - - - - - - - - - - - - - - - - forwarded by vince j kaminski / hou / ect on 04 / 10 / 2001  03 : 02 pm - -
- - - - - - - - - - - - - - - - - - - - - - - - -  stinson gibner  04 / 10 / 2001 02 : 57 pm  to : vince j kaminski /
hou / ect @ ect  cc :  subject : sevil yamin  vince ,  do you want me to do this , or vasant ?  - - stinson  - - - - -
- - - - - - - - - - - - - - - - - forwarded by stinson gibner / hou / ect on 04 / 10 / 2001  02 : 57 pm - - - - - - - -
- - - - - - - - - - - - - - - - - - -  from : anne labbe / enron @ enronxgate on 04 / 06 / 2001 09 : 57 am  to :
stinson gibner / hou / ect @ ect  cc :  subject : sevil yamin  stinson ,  i am the new hr generalist for the research
group because norma villarreal is  moving to business analysis and reporting . earlier this week , norma and i  met
with vince , and he said that he was going to talk to you about writing up  a list of sevil ' s projects /
accomplishments for last year and this year , so  that we can give her a project bonus since she did not receive a
bonus during  the normal prc time . at your earliest convenience , will you please email me  this list so that i can
get started putting together all of the paperwork so  that she can receive a check on april 16 th .  if you have any
questions , please feel free to contact me at 5 - 7809 . i look  forward to meeting you , and the rest of the group
next week at vince ' s staff  meeting .  thanks ,  anne labbe '",
       0], dtype=object)
```

```
sp = spacy.load('en_core_web_sm')
```

```
text, label = emails_raw[0]
print(text)
document = sp(text.lower())
print(document)
print(type(document))
```

```
Subject: sevil yamin  anne ,  vasant sent this information to norma . i shall fwd his message to you .  vince  - - - - -
subject: sevil yamin  anne ,  vasant sent this information to norma . i shall fwd his message to you .  vince  - - - - -
<class 'spacy.tokens.doc.Doc'>
```

```
for word in document:
    print(word)
    print(type(word))
    print(str(word))
    print(type(str(word)))
    break
```

```
subject
<class 'spacy.tokens.token.Token'>
subject
<class 'str'>
```

```
model[str(word)]
```

```
array([-0.098252 ,  0.053359 ,  0.3814   ,  0.25006  ,  0.37622  ,
        0.39795  , -0.42648  , -0.11512  , -0.062542 , -0.30872  ,
       -0.1126   ,  0.017485 ,  0.42269  , -0.05028  , -0.036123 ,
       -0.17231  ,  0.022046 ,  0.269    , -0.23359  ,  0.012059 ,
       -0.17243  , -0.204    , -0.0737   , -0.11341  ,  0.11035  ,
       -0.56162  ,  0.080746 , -0.36331  , -0.30045  , -0.12815  ,
       -0.45184  ,  0.012891 , -0.1789   , -0.44184  ,  0.093577 ,
        0.59694  ,  0.023494 , -0.4611   , -0.45794  ,  0.11821  ,
       -0.5165   , -0.11584  ,  0.083922 ,  0.016293 , -0.25888  ,
       -0.30356  ,  0.28238  , -0.031084 ,  0.045921 , -0.65211  ,
        0.98727  ,  0.023694 ,  0.22667  ,  0.75697  , -0.20568  ,
       -1.3769   ,  0.41452  , -0.54832  ,  1.5247   ,  0.16025  ,
        0.35099  ,  0.99004  , -0.32081  , -0.51617  ,  1.7852   ,
       -0.36472  ,  0.52039  , -0.0099243,  0.066323 ,  0.073432 ,
       -0.40247  ,  0.04315  ,  0.49818  ,  0.50381  , -0.30446  ,
       -0.10126  , -0.33538  , -0.44573  , -1.0383   , -0.37519  ,
        0.32278  , -0.35961  ,  0.15101  , -0.10449  , -1.7589   ,
        0.0035028, -0.3349   , -0.47833  ,  0.37978  , -0.68465  ,
       -0.27228  , -0.041006 , -0.028524 ,  0.13495  ,  0.28592  ,
        0.7695   , -0.046894 , -0.42496  ,  0.20771  , -0.003625 ],
      dtype=float32)
```

```
emails_raw.shape
```

```
(28138, 2)
```

```
for email in tqdm(emails_raw):
    text, label = email
    print(text)
    break
```

```
  0%|          | 0/28138 [00:00<?, ?it/s]Subject: sevil yamin  anne ,  vasant sent this information to norma . i shall f
```

```
# emails_embeddings = torch.ones(100).to(device)
# emails_embeddings = np.zeros(100)
emails_embeddings = [] #This will be a list of np arrays
# print(emails_embeddings)
count = 0
```

```
for email in tqdm(emails_raw):
    text, label = email
    document = sp(text.lower())

    words_in_doc = 0
    # doc_embedding = torch.zeros(100).to(device)
    doc_embedding = np.zeros(100)


    for word in document:

        if str(word)in model:
            words_in_doc+=1
            word_embedding = model[str(word)]
            # torch_word_embedding = torch.from_numpy(word_embedding).to(device)
            doc_embedding+=word_embedding




    doc_embedding = doc_embedding/words_in_doc
    # print(doc_embedding)
    # print(doc_embedding.shape)
    # emails_embeddings = torch.cat((emails_embeddings, doc_embedding),dim=0)
    # print(emails_embeddings)
    emails_embeddings.append(doc_embedding)
    # print(emails_embeddings)
    # print(len(emails_embeddings))
    # count+=1
    # if count==2:
    #   break
```

```
 15%|█         | 4096/28138 [03:30<20:37, 19.43it/s]
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-21-28b0a0e0ad1b> in <cell line: 6>()
     17         if str(word)in model:
     18             words_in_doc+=1
---> 19             word_embedding = model[str(word)]
     20             # torch_word_embedding = torch.from_numpy(word_embedding).to(device)
     21             doc_embedding+=word_embedding

/usr/local/lib/python3.10/dist-packages/gensim/models/keyedvectors.py in __getitem__(self, key_or_keys)
    401         """
    402         if isinstance(key_or_keys, _KEY_TYPES):
--> 403             return self.get_vector(key_or_keys)
    404 
    405         return vstack([self.get_vector(key) for key in key_or_keys])

KeyboardInterrupt: 
```

EXPLAIN ERROR

Start coding or generate with AI.

```
device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
emails_embeddings[0]
```

```
array([-4.37622214e-01,  2.91747647e-01,  4.46433682e-01, -4.99543788e-01,
       -1.55564459e-01,  5.86799765e-02,  1.51524022e-01,  6.33064256e-02,
       -2.04417315e-01,  1.78659207e-01,  3.04671649e-01, -1.05319386e-01,
        1.21534547e-01,  1.48667727e-01,  2.95478057e-01, -6.14197867e-02,
        2.78467246e-01, -3.19167350e-01, -3.50274674e-01,  1.02720839e-01,
        3.29825392e-01,  2.67235508e-01,  1.35532020e-01,  4.88986030e-01,
        3.69819844e-01,  8.62888053e-02, -8.14431650e-02, -5.01819413e-01,
       -1.05849846e-01, -1.97657797e-01, -1.17333600e-01,  3.18218639e-01,
        7.71797097e-02,  1.21786967e-01, -7.53295493e-02,  3.55375170e-02,
        5.41540271e-02,  4.01104181e-01, -1.44676295e-01, -6.71737383e-02,
       -2.18747586e-01, -5.05502406e-01, -2.97167646e-03, -4.22863999e-01,
        3.31642116e-03, -8.27475752e-02, -2.41133589e-01, -5.72058499e-01,
        2.18677925e-01, -6.74991009e-01, -2.76503819e-01, -1.02604158e-01,
       -1.03371694e-03,  8.80221728e-01, -5.07553200e-01, -2.11793263e+00,
        1.33619580e-01,  2.80513532e-01,  1.56657101e+00,  3.23621667e-01,
       -5.43159460e-02,  4.53058211e-01, -1.42964855e-01, -1.01533485e-01,
        4.52465011e-01,  1.38532799e-01,  6.42140523e-01,  4.92330105e-01,
       -1.06035340e-01, -8.84444897e-03,  1.21807130e-01, -1.95119042e-01,
       -3.67987198e-01, -6.28593519e-01, -2.84834603e-02, -1.40871496e-01,
        2.24763613e-02, -1.57289068e-01, -9.48109343e-01, -3.32549420e-01,
        6.15616773e-01, -2.25693989e-01, -8.79166591e-02,  1.34347621e-02,
       -1.23637051e+00, -3.48560098e-03, -4.43250373e-03, -9.82094594e-02,
       -3.96876909e-02, -2.76937921e-01, -6.56716241e-02,  9.63152081e-02,
       5.07243956e-02,  1.29780755e-01, -4.10787589e-01,  9.61688370e-02,
       2.39604914e-02, -6.34066986e-01,  4.61695212e-01, -1.32738828e-01])
```

```
len(emails_embeddings)
```

```
28138
```

```
emails_embeddings[0].shape
```

```
(100,)
```

```
import csv
```

```
# with open('emails_embeddings.csv', mode='w', newline='') as file:
#     writer = csv.writer(file)
#     writer.writerows(emails_embeddings)
```

```
# emails_embeddings_nparray = np.array(emails_embeddings)
```

```
# file_name = 'emails_embeddings_np.csv'

# # Use the 'savetxt' function from numpy
# np.savetxt(file_name, emails_embeddings, delimiter=',')
```

```
from torch.utils.data import DataLoader
from torchtext.data.functional import to_map_style_dataset
```

```
count = 0
y = []
for email in tqdm(emails_raw):
    text,label = email
    # print(label)
    y.append(label)
    # print(label.shape)
    # count+=1
    # if count==5:
    #   break
```

```
100%|██████████| 28138/28138 [00:00<00:00, 697554.97it/s]
```

```
y = np.array(y)
```

```
y.shape
```

```
(28138,)
```

```
y
```

```
array([0, 1, 0, ..., 1, 0, 1])
```

```
y[0]
```

```
0
```

```
# divide into training, validation, and testing data sets

def separate_data(X,Y,percent_validation=0.1,percent_test=0.1):

    N = len(X)
    len_validation = int(percent_validation*N)
    len_test = int(percent_test*N)
    len_train = N - len_validation - len_test

    len_t_v = len_train+len_validation

    X_train      = X[:len_train]
    Y_train      = Y[:len_train]
    X_validation = X[len_train:len_t_v]
    Y_validation = Y[len_train:len_t_v]
    X_test = X[len_t_v:]
    Y_test = Y[len_t_v:]

    return (X_train,Y_train,X_validation,Y_validation,X_test,Y_test)
```

```
# tests

# X_ = np.array(list(range(100)))
# Y_ = np.array(list(range(100)))

# separate_data(X_,Y_)
# separate_data(X_,Y_,0.2,0.2)


X_ = torch.tensor(emails_embeddings,dtype=torch.float32)
Y_ = torch.tensor(y, dtype=torch.long)
```

```
X_.shape
```

```
torch.Size([28138, 100])
```

```
Y_.shape
```

```
torch.Size([28138])
```

```
print(X_.size(),'\n')
print(Y_.size(),'\n')
# print(X_[:5],'\n')
# print(Y_[:5],'\n')

print('     Point Coordinates    ','\tGroup Label')
print('--------------------------','\t-----------')
# for k in range(10):
#     print(f'{X_[k]}\t    {Y_[k]}')
```

```
torch.Size([28138, 100])

torch.Size([28138])

     Point Coordinates          Group Label
--------------------------      -----------
```

```
X_train_blobs,Y_train_blobs,X_val_blobs,Y_val_blobs,X_test_blobs,Y_test_blobs = \
                                                                        separate_data(X_,Y_,0.1,0.1)

N_train,N_val,N_test = len(X_train_blobs),len(X_val_blobs),len(X_test_blobs)
```

```
print('X_train_blobs.size():',X_train_blobs.size())
print('Y_train_blobs.size():',Y_train_blobs.size())
print('X_val_blobs.size():  ',X_val_blobs.size())
print('Y_val_blobs.size():  ',Y_val_blobs.size())
print('X_test_blobs.size(): ',X_test_blobs.size())
print('Y_test_blobs.size(): ',Y_test_blobs.size())
print(X_test_blobs[0], Y_test_blobs[0])
```

```
X_train_blobs.size(): torch.Size([22512, 100])
Y_train_blobs.size(): torch.Size([22512])
X_val_blobs.size():   torch.Size([2813, 100])
Y_val_blobs.size():   torch.Size([2813])
X_test_blobs.size():  torch.Size([2813, 100])
Y_test_blobs.size():  torch.Size([2813])
tensor([-0.2927,  0.2425,  0.4053, -0.3239, -0.1126,  0.0991,  0.0123,  0.0287,
        -0.0483,  0.0437,  0.3799, -0.1402,  0.0832,  0.1118,  0.2205, -0.2183,
         0.1848, -0.0485, -0.3297,  0.2099,  0.1884,  0.0321,  0.1937,  0.2619,
         0.2216,  0.0816,  0.0558, -0.2924,  0.0272, -0.2180, -0.0038,  0.4076,
        -0.0699,  0.0327, -0.0356,  0.2110,  0.0848,  0.2575, -0.0182, -0.0851,
        -0.0865, -0.3785, -0.0178, -0.4077, -0.0760, -0.1261, -0.0393, -0.3777,
        -0.0033, -0.5843,  0.0348, -0.0375,  0.0372,  0.7302, -0.4406, -1.8577,
         0.0094, -0.0559,  1.2972,  0.4387, -0.2408,  0.3734, -0.1913, -0.0196,
         0.6219,  0.0048,  0.3600,  0.3556,  0.3466, -0.0067, -0.1451, -0.1450,
        -0.1582, -0.3864,  0.1154, -0.0439,  0.0084, -0.0239, -0.8111, -0.3025,
         0.5421, -0.1936, -0.2635, -0.0303, -0.9652, -0.0094,  0.0668, -0.2516,
         0.0219, -0.2110, -0.0899,  0.0232, -0.0150,  0.1343, -0.3866,  0.0461,
         0.0045, -0.3458,  0.3420, -0.1151]) tensor(0)
```

```
class SpamModel(torch.nn.Module):

        # We first define a number of local variables for layers

        def __init__(self):
            super(SpamModel,self).__init__()
            self.linear1 = torch.nn.Linear(100,15)
            self.activation1 = torch.nn.ReLU()
            self.linear2 = torch.nn.Linear(15,2)
        # foward defines the forward pass of a FFNN,
        # sending a vector x through each layer and then returning it

        def forward(self,x):

            x = self.linear1(x)
            x = self.activation1(x)                  # we have to explicitly send x through the sigmoid function
            x = self.linear2(x)
                                                  # we will take care of the softmax later
            return x


spam_model = SpamModel()

print(spam_model)
```

```
SpamModel(
  (linear1): Linear(in_features=100, out_features=15, bias=True)
  (activation1): ReLU()
  (linear2): Linear(in_features=15, out_features=2, bias=True)
)
```

```
# num_epoches = 100

# loss_fn = nn.CrossEntropyLoss()

# # optimizer = optim.SGD(blobs_model.parameters(),lr=0.1)
# # optimizer = optim.Adam(blobs_model.parameters())
# optimizer = optim.Adagrad(spam_model.parameters(),lr =.01)
# # optimizer = optim.RMSprop(blobs_model.parameters())

# num_epochs = num_epoches

# for epoch in tqdm(range(num_epochs)):           # for each epoch

#     spam_model.train()     # We are in training mode, so keep track of differentials for backtracking

#     for k in range(len(X_train_blobs)):         # train the network on all the training data

#         optimizer.zero_grad                     # reset the differentials to 0

#         Y_hat = spam_model(X_train_blobs[k])      # forward pass for this batch; Y_hat is output of network

#         loss = loss_fn(Y_hat,Y_train_blobs[k])     # calculate the loss

#         loss.backward()                         # do backpropagation to calculate the differentials

#         optimizer.step()                        # adjust parameters based on one step of gradient descent
```

```
100%|██████████| 100/100 [23:53<00:00, 14.34s/it]
```

```
# blobs_model(X_test_blobs[0])
```

```
tensor([-0.2449, -0.2283], grad_fn=<ViewBackward0>)
```

```
# sm = nn.Softmax(dim=0)

# def getPrediction(x):
#     return torch.argmax(x).item()

# for k in range(20):
#     Y_hat = getPrediction(spam_model(X_train_blobs[k]))
#     print(f'{X_test_blobs[k]}\t{Y_test_blobs[k]}   {Y_hat}   {sm(spam_model(X_test_blobs[k]))}')
```

```
tensor([-0.2927,  0.2425,  0.4053, -0.3239, -0.1126,  0.0991,  0.0123,  0.0287,
        -0.0483,  0.0437,  0.3799, -0.1402,  0.0832,  0.1118,  0.2205, -0.2183,
         0.1848, -0.0485, -0.3297,  0.2099,  0.1884,  0.0321,  0.1937,  0.2619,
         0.2216,  0.0816,  0.0558, -0.2924,  0.0272, -0.2180, -0.0038,  0.4076,
        -0.0699,  0.0327, -0.0356,  0.2110,  0.0848,  0.2575, -0.0182, -0.0851,
        -0.0865, -0.3785, -0.0178, -0.4077, -0.0760, -0.1261, -0.0393, -0.3777,
        -0.0033, -0.5843,  0.0348, -0.0375,  0.0372,  0.7302, -0.4406, -1.8577,
         0.0094, -0.0559,  1.2972,  0.4387, -0.2408,  0.3734, -0.1913, -0.0196,
         0.6219,  0.0048,  0.3600,  0.3556,  0.3466, -0.0067, -0.1451, -0.1450,
        -0.1582, -0.3864,  0.1154, -0.0439,  0.0084, -0.0239, -0.8111, -0.3025,
         0.5421, -0.1936, -0.2635, -0.0303, -0.9652, -0.0094,  0.0668, -0.2516,
         0.0219, -0.2110, -0.0899,  0.0232, -0.0150,  0.1343, -0.3866,  0.0461,
         0.0045, -0.3458,  0.3420, -0.1151])    0   0   tensor([0.8741, 0.1259], grad_fn=<SoftmaxBackward0>)
tensor([-1.3047e-01,  2.0481e-01,  3.5641e-01, -2.1388e-01, -4.6393e-02,
         9.9975e-02, -4.5502e-02,  1.1489e-01, -1.6490e-02, -7.7640e-03,
         2.4743e-01, -3.9291e-02,  1.3479e-01,  5.5976e-02,  1.7461e-01,
        -1.9807e-01,  1.7694e-01,  1.0315e-01, -3.5122e-01,  1.8643e-01,
         1.2270e-01, -1.0591e-01,  1.4454e-01,  2.3331e-01,  1.6104e-01,
        -2.5166e-02,  7.6484e-02, -2.8263e-01,  1.5986e-02, -2.1452e-01,
            -1.4677e-02,  4.1550e-01, -1.4443e-01,  4.6209e-02,  3.6491e-02,
             2.5571e-01,  5.0622e-02,  1.5719e-01,  1.9666e-02, -1.0918e-01,
            -1.7211e-01, -2.2560e-01, -5.4742e-03, -2.8661e-01, -4.5631e-02,
            -4.0476e-02, -3.4668e-03, -2.8247e-01, -5.4806e-02, -5.1350e-01,
             7.5047e-02,  8.4152e-02,  9.2686e-03,  7.3280e-01, -2.9956e-01,
            -1.7244e+00, -3.0908e-02, -1.4853e-01,  1.2655e+00,  2.8445e-01,
            -1.5402e-01,  4.6025e-01, -1.7933e-01,  7.6048e-02,  5.2014e-01,
             2.5976e-02,  2.3911e-01,  1.7478e-01,  2.5239e-01, -1.2637e-01,
            -4.7380e-02, -1.4795e-01, -6.8020e-03, -2.3925e-01,  1.4245e-01,
             1.5731e-03, -7.2940e-02,  5.6998e-02, -6.8331e-01, -1.3890e-01,
             4.1057e-01, -5.6814e-02, -3.6894e-01,  1.4045e-01, -9.5796e-01,
            -6.4837e-02,  1.0585e-01, -1.7833e-01, -4.2315e-02, -1.4301e-01,
            -6.7567e-02, -8.3640e-03, -5.5069e-03,  6.9822e-02, -3.9658e-01,
            -3.0321e-02, -8.5653e-02, -2.7335e-01,  4.2589e-01,  8.3183e-02])       1   1   tensor([0.0227, 0.9773], grad_fn
    tensor([-0.1394,  0.1557,  0.2127, -0.1863, -0.0711,  0.1016,  0.0466,  0.0697,
            -0.0626,  0.0635,  0.0639, -0.0383,  0.0327,  0.0102,  0.1771, -0.0654,
             0.1580,  0.0162, -0.1036,  0.1192,  0.1161,  0.0140,  0.0976,  0.1601,
             0.2481,  0.0943, -0.0563, -0.1786, -0.0973, -0.1489, -0.0185,  0.1467,
            -0.0048,  0.0352, -0.0328,  0.1082,  0.0331,  0.1298,  0.0585, -0.1473,
            -0.0352, -0.1492, -0.0480, -0.2032, -0.0459, -0.0260,  0.0138, -0.2102,
             0.0615, -0.2945, -0.1374, -0.0662,  0.0248,  0.4778, -0.2644, -1.1869,
             0.0514, -0.0296,  0.7588,  0.1506, -0.0564,  0.3216, -0.0688,  0.0519,
             0.2507, -0.0355,  0.3463,  0.1252,  0.0612, -0.1170, -0.0019, -0.1353,
            -0.0948, -0.1586, -0.0135, -0.0416, -0.0212, -0.0782, -0.4376, -0.0752,
             0.2882, -0.0339, -0.1353,  0.0912, -0.6061, -0.0259,  0.0409, -0.0179,
            -0.0150, -0.1327, -0.0279,  0.0106, -0.0102,  0.0637, -0.2281, -0.0654,
            -0.0457, -0.2454,  0.2806,  0.0277])    1   0   tensor([0.0283, 0.9717], grad_fn=<SoftmaxBackward0>)
    tensor([-1.5283e-01,  2.5651e-01,  4.2226e-01, -3.3095e-01, -1.0451e-01,
             7.2406e-02, -1.7481e-02,  1.9463e-01, -1.3313e-01, -9.0454e-02,
             3.8479e-01, -2.0768e-01,  8.9644e-02,  1.4485e-01,  1.3097e-01,
            -1.4732e-01,  1.9016e-01,  8.6533e-02, -4.6516e-01,  3.8378e-01,
             1.7875e-01,  3.6282e-02,  1.6899e-01,  2.8125e-01,  1.9059e-01,
             1.0812e-01,  1.2116e-01, -3.2931e-01,  1.6612e-01, -1.3701e-01,
            -1.4590e-01,  4.9951e-01, -2.6884e-02,  5.2437e-02, -4.8672e-02,
             2.6107e-01,  6.8760e-02,  2.8029e-01,  2.2093e-02, -1.0819e-01,
            -2.0548e-01, -2.7846e-01, -3.1808e-02, -3.5428e-01, -9.6304e-02,
            -8.9178e-02, -1.1880e-01, -3.9833e-01, -1.8644e-02, -5.0160e-01,
             4.8188e-03, -5.0103e-02,  1.9728e-02,  8.1532e-01, -3.6268e-01,
            -1.9958e+00,  3.1314e-04, -2.9233e-02,  1.1182e+00,  3.4517e-01,
```

```
# blobs_model = BlobsModel()

# # print(blobs_model)

# loss_fn = nn.CrossEntropyLoss()

# # optimizer = optim.SGD(blobs_model.parameters(),lr=0.0001)
# # optimizer = optim.Adam(blobs_model.parameters())
# optimizer = optim.Adagrad(blobs_model.parameters(),lr=0.01)
# # optimizer = optim.RMSprop(blobs_model.parameters(),lr=0.001)

# num_epochs = 100

# for epoch in tqdm(range(num_epochs)):           # for each epoch

#     blobs_model.train()                     # We are in training mode, so keep track of differentials for backtracking

#     for k in range(len(X_train_blobs)):         # train the network on all the training data

#         optimizer.zero_grad                     # reset the differentials to 0

#         Y_hat = blobs_model(X_train_blobs[k])      # forward pass for this batch; Y_hat is output of network

#         loss = loss_fn(Y_hat,Y_train_blobs[k])     # calculate the loss

#         loss.backward()                         # do backpropagation to calculate the differentials

#         optimizer.step()                        # adjust parameters based on one step of gradient descent


# blobs_model.eval()          # stop doing backprob, just run model as feed-forward phase
# num_correct_test = 0

# for k in range(len(X_test_blobs)):

#     Y_hat_test = blobs_model(X_test_blobs[k])
#     if getPrediction(Y_hat_test) == Y_test_blobs[k]:
#         num_correct_test += 1

# test_accuracy = num_correct_test / len(X_test_blobs)

# print("Accuracy:", test_accuracy )


# blobs_model.eval()          # stop doing backprob, just run model as feed-forward phase
# num_correct_test = 0
# for k in range(len(X_test_blobs)):

#     Y_hat_test = blobs_model(X_test_blobs[k])
#     if getPrediction(Y_hat_test) == Y_test_blobs[k]:
#         num_correct_test += 1

# test_accuracy = num_correct_test / len(X_test_blobs)

# print("Accuracy:", test_accuracy )
```

```
Accuracy: 0.5879843583362958
```

```
from sklearn.datasets import make_blobs
from torch.utils.data import Dataset
from torch.utils.data import DataLoader


class BlobsDataset(Dataset):

    def __init__(self, X,Y):
        self.X_blobs = X
        self.Y_blobs = Y

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

    # return a pair x,y at the index idx in the data set
    def __getitem__(self, idx):
        return self.X_blobs[idx], self.Y_blobs[idx]


batch_size = 64

blobs_ds = BlobsDataset(X_,Y_)

gen = torch.Generator().manual_seed(0)  # this will ensure the same split every time

train_blobs_ds,val_blobs_ds,test_blobs_ds = random_split(blobs_ds, [0.8,0.1,0.1], generator=gen)

blobs_training_dataloader =   DataLoader(train_blobs_ds, batch_size=batch_size, shuffle=True)

blobs_validation_dataloader = DataLoader(val_blobs_ds,   batch_size=batch_size, shuffle=True)

blobs_testing_dataloader =    DataLoader(test_blobs_ds,  batch_size=batch_size, shuffle=True)
```

```
for x,y in blobs_training_dataloader:
    print(x)
    print(y)
    break
```

```
tensor([[-0.1760,  0.2254,  0.5354,  ..., -0.3825,  0.5371,  0.2332],
        [-0.1530,  0.3672,  0.4590,  ..., -0.3677,  0.5838,  0.0286],
        [-0.1202,  0.2702,  0.3504,  ..., -0.3594,  0.4407, -0.0018],
        ...,
        [-0.1263,  0.1172,  0.3661,  ..., -0.4050,  0.4628,  0.2197],
        [-0.3458,  0.3118,  0.4597,  ..., -0.4296,  0.4174,  0.0276],
        [-0.1099,  0.2412,  0.3736,  ..., -0.2650,  0.5147,  0.1323]])
tensor([0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
        0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0])
```

```
for x,y in blobs_training_dataloader:
    print(x.shape,y.shape)
    break
```

```
torch.Size([64, 100]) torch.Size([64])
```

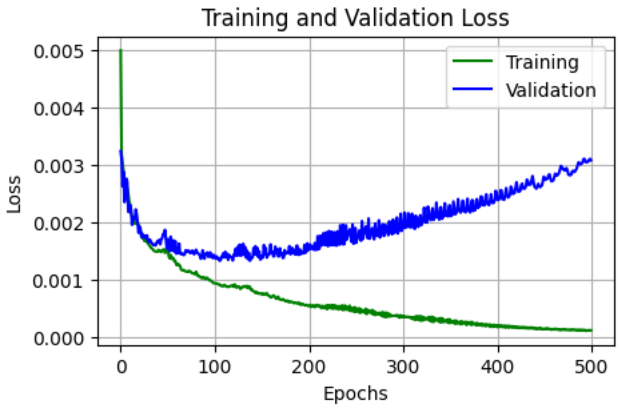
```
def show_performance_curves(training_loss,validation_loss,training_accuracy,validation_accuracy,test_accuracy):

    plt.figure(figsize=(5, 3))
    plt.plot(training_loss,label='Training',color='g')
    plt.plot(validation_loss,label='Validation',color='b')
    plt.title('Training and Validation Loss')
    plt.legend(loc='upper right')
#     plt.ylim(-0.1,(max(max(training_loss),max(validation_loss))*1.1) )
    plt.grid()
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.show()

    print('Final Training Loss:  ', np.around(training_loss[-1],6))
    print('Final Training Loss:  ',np.around(training_loss[-1],6))
    print('Final Validation Loss:',np.around(validation_loss[-1],6))

    plt.figure(figsize=(5, 3))
    plt.plot(training_accuracy,label='Training',color='g')
    plt.plot(validation_accuracy,label='Validation',color='b')
    plt.title('Training and Validation Accuracy')
    plt.legend(loc='lower right')
#     plt.ylim(-0.1,1.1)
    plt.grid()
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.show()

    print('Final Training Accuracy:  ',np.around(training_accuracy[-1],6))
    print('Final Validation Accuracy:',np.around(validation_accuracy[-1],6))
    print()

    print("Test Accuracy:", test_accuracy.item())
    print()


blobs_model2 = BlobsModel()

# print(blobs_model)

loss_fn = nn.CrossEntropyLoss()
# optimizer = optim.SGD(blobs_model2.parameters(),lr=0.001)
# optimizer = optim.Adam(blobs_model2.parameters())
optimizer = optim.Adagrad(blobs_model2.parameters(),lr=0.01)
# optimizer = optim.RMSprop(blobs_model2.parameters(),lr=0.001)

num_epochs = 500

batch_size = 64

training_losses = np.zeros(num_epochs)
val_losses      = np.zeros(num_epochs)

training_accuracy = np.zeros(num_epochs)
val_accuracy      = np.zeros(num_epochs)

for epoch in tqdm(range(num_epochs)):
    # training
    blobs_model2.train()
    t_loss = 0.0
    t_num_correct = 0

    for X_train_batch,Y_train_batch in blobs_training_dataloader:

        optimizer.zero_grad
        Y_train_hat = blobs_model2(X_train_batch)
        loss = loss_fn(Y_train_hat,Y_train_batch)
        loss.backward()
        optimizer.step()
        t_loss += loss.item()

        # If we just use the scalar class number, it must be a long (we did this when creating the dataset)
        t_num_correct += (torch.argmax(Y_train_hat,dim=1) == Y_train_batch).float().sum()

    training_losses[epoch]   = t_loss/N_train
    training_accuracy[epoch] = t_num_correct/N_train

    #  validation
    v_loss = 0.0
    blobs_model2.eval()
    v_num_correct = 0

    for X_val_batch,Y_val_batch in blobs_validation_dataloader:

        Y_hat_val = blobs_model2(X_val_batch)
        loss = loss_fn(Y_hat_val,Y_val_batch)
        v_loss += loss.item()

        v_num_correct += (torch.argmax(Y_hat_val,dim=1) == Y_val_batch).float().sum()

    val_losses[epoch]   = v_loss/N_val
    val_accuracy[epoch] = v_num_correct/N_val

#  testing
num_correct_test = 0
blobs_model2.eval()
```

```
for X_test_batch,Y_test_batch in blobs_testing_dataloader:
    Y_hat_test = blobs_model2(X_test_batch)
    num_correct_test += (torch.argmax(Y_hat_test,dim=1) == Y_test_batch).float().sum()

test_accuracy = num_correct_test / N_test

show_performance_curves(training_losses,val_losses,training_accuracy,val_accuracy,test_accuracy)
```

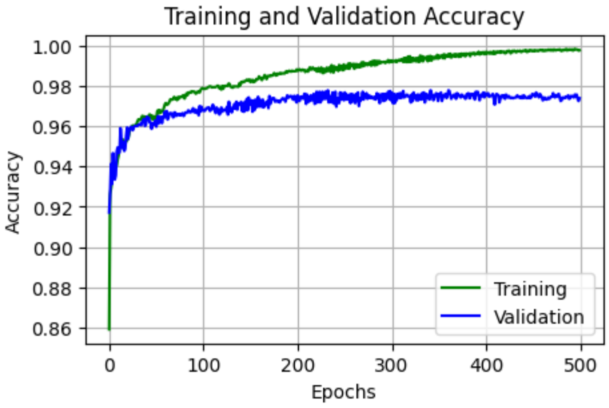
```
100%|██████████| 500/500 [04:06<00:00,  2.03it/s]
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-72-7dfe4991b3c2> in <cell line: 68>()
     66 test_accuracy = num_correct_test / N_test
     67
---> 68 show_performance_curves(training_losses,val_losses,training_accuracy,val_accuracy,test_accuracy)

NameError: name 'show_performance_curves' is not defined
```

SEARCH STACK OVERFLOW

```
show_performance_curves(training_losses,val_losses,training_accuracy,val_accuracy,test_accuracy)
```



```
Final Training Loss:   0.000124
Final Validation Loss: 0.003089
```



```
Final Training Accuracy:   0.997868
Final Validation Accuracy: 0.974049

Test Accuracy: 0.9687166810035706
```