harikrishnadevhw1-pytorch

August 28, 2023

1 HW1 (15 Points):

Required Submissions:

You have to submit two files for this part of the HW 1. FirstNameLastName_Hw1a.ipynb (colab notebook) 2. FirstNameLastName_Hw1a.pdf pdf file.

The pdf file is just the pdf version of colab notebook.

[11]:
```
import torch
import time
import numpy as np
import matplotlib.pyplot as plt
```

[12]:
```
torch.set_printoptions(precision=4, sci_mode=False)
```

Q1 : Create Tensor (1 Point) Create a torch Tensor of shape (5, 3) which is filled with zeros. Modify the tensor to set element (0, 2) to 10 and element (2, 0) to 100.

[13]:
```
tensor = torch.zeros((5, 3))

tensor[0, 2] = 10
tensor[2, 0] = 100

print(tensor)
```

```
tensor([[  0.,   0.,  10.],
        [  0.,   0.,   0.],
        [100.,   0.,   0.],
        [  0.,   0.,   0.],
        [  0.,   0.,   0.]])
```

2 Q2: Reshape tensor (1 Point)

You have following tensor as input:

`x=torch.tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23])`

Using only reshaping functions (like view, reshape, transpose, permute), you need to get at the following tensor as output:

```
tensor([[ 0,  4,  8, 12, 16, 20],
        [ 1,  5,  9, 13, 17, 21],
        [ 2,  6, 10, 14, 18, 22],
        [ 3,  7, 11, 15, 19, 23]])
```

[14]:
```
x=torch.tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
 ↪18, 19, 20, 21, 22, 23])
```

[15]:
```
reshaped = x.view(6, 4)
output = reshaped.transpose(0, 1)
print(output)
```

```
tensor([[ 0,  4,  8, 12, 16, 20],
        [ 1,  5,  9, 13, 17, 21],
        [ 2,  6, 10, 14, 18, 22],
        [ 3,  7, 11, 15, 19, 23]])
```

3 Question 3. Speedtest for vectorization - 2 Points

Your goal is to measure the speed of linear algebra operations for different levels of vectorization.

1. Construct two matrices A and B with Gaussian random entries of size 1024×1024.
2. Compute $C = AB$ using matrix-matrix operations and report the time. (Hint: Use torch.mm)
3. Compute $C = AB$, treating A as a matrix but computing the result for each column of B one at a time. Report the time. (hint use torch.mv inside a for loop)
4. Compute $C = AB$, treating A and B as collections of vectors. Report the time. (Hint: use torch.dot inside nested for loop)

[16]:
```
## Solution 1
torch.manual_seed(42) # dod not chnage this
A = torch.randn(1024, 1024)
B = torch.randn(1024, 1024)
```

[17]:
```
## Solution 2
start=time.time()

C = torch.mm(A, B)

print("Matrix by matrix: " + str(time.time()-start) + " seconds")
```

```
Matrix by matrix: 0.05514264106750488 seconds
```

[18]:
```
## Solution 3
C= torch.empty(1024,1024)
start = time.time()
```

```
for i in range(1024):
    C[:, i] = torch.mv(A, B[:, i])

print("Matrix by vector: " + str(time.time()-start) + " seconds")
```

```
Matrix by vector: 0.29326939582824707 seconds
```

[19]:
```
## Solution 4
C= torch.empty(1024,1024)
start = time.time()

for i in range(1024):
    for j in range(1024):
        C[i, j] = torch.dot(A[i], B[j])

print("vector by vector: " + str(time.time()-start) + " seconds")
```

```
vector by vector: 15.7226083278656 seconds
```

4 Question 4 : Redo Question 3 by using GPU - 2 Point

Using GPUs

How to use GPUs in Google Colab In Google Colab – Go to Runtime Tab at top – select change runtime type – for hardware accelartor choose GPU

[20]:
```
# Check if GPU is availaible
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda:0
```

[21]:
```
## Solution 1
torch.manual_seed(42)
A= torch.randn((1024, 1024),device=device)
B= torch.randn((1024, 1024),device=device)
```

[22]:
```
## Solution 2
start=time.time()

C = torch.mm(A, B)

print("Matrix by matrix: " + str(time.time()-start) + " seconds")
```

```
Matrix by matrix: 2.789036273956299 seconds
```

[23]:
```
## Solution 3
C= torch.empty(1024,1024, device = device)
```

```
start = time.time()

for i in range(1024):
    C[:, i] = torch.mv(A, B[:, i])

print("Matrix by vector: " + str(time.time()-start) + " seconds")
```

```
Matrix by vector: 0.05862569808959961 seconds
```

[24]:
```
## Solution 4
C= torch.empty(1024,1024, device = device)
start = time.time()

for i in range(1024):
    for j in range(1024):
        C[i, j] = torch.dot(A[i], B[j])

print("vector by vector: " + str(time.time()-start) + " seconds")
```

```
vector by vector: 36.108577728271484 seconds
```

5 Question 5. Memory efficient computation - 2 Points

We want to compute $C \leftarrow A \cdot B + C$, where A, B and C are all matrices. Implement this in the most memory efficient manner. Pay attention to the following two things:

1. Do not allocate new memory for the new value of C.
2. Do not allocate new memory for intermediate results if possible. Hint: If you implement this correcly the memory location of C given by id(C) will be same in both the cells below.

[25]:
```
A= torch.randn((1000, 1000),device=device)
B= torch.randn((1000,1000),device=device)
C= torch.randn((1000, 1000),device=device)
print(id(C))
```

```
135463344210064
```

[26]:
```
C.addmm_(A, B)

print(id(C))
```

```
135463344210064
```

#Question 6. Broadcast Operations - 2 Points

In order to perform polynomial fitting we want to compute a design matrix A with

$$A_{ij} = x_i^j$$

Our goal is to implement this **without a single for loop** entirely using vectorization and broadcast. Here $1 \leq j \leq 3$ and $x = \{1, 2, 3, 4, 5\}$. Implement code that generates following A matrix

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 8 \\ 3 & 9 & 27 \\ 4 & 16 & 64 \\ 5 & 25 & 125 \end{bmatrix}$$

[27]:
```
x = torch.tensor([1, 2, 3, 4, 5], dtype=torch.float32)
powers = torch.arange(1, 4, dtype=torch.float32)
x_powers = x.view(-1,1) ** powers
print(x_powers)
```

```
tensor([[  1.,   1.,   1.],
        [  2.,   4.,   8.],
        [  3.,   9.,  27.],
        [  4.,  16.,  64.],
        [  5.,  25., 125.]])
```

6 Q7 Image Classification using - Pixel Similarity - 5 Points

6.1 Import libraries

[28]:
```
# import libraries
import torchvision
import torchvision.transforms as transforms
from pathlib import Path
import cv2 as cv
from google.colab.patches import cv2_imshow # for image display
import pandas as pd
```

6.2 Mount Google Deive

[29]:
```
# mount google drive
from google.colab import drive
drive.mount('/content/drive')
```

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

6.3 Path to Dowanload Data

Make sure you change the Path to where you want to save data. In the code below - **data/datasets** is the folder name in my google drive. You can change this to appropriate folder for your drive. For example you may want to save data to BUAN6341/HW1/Data. In this case the below code should be modified to :

data_folder = Path('/content/drive/MyDrive/BUAN6341/HW2/Data')

[30]:
```
data_folder = Path('/content/drive/MyDrive/Colab Notebooks/BUAN 6382/Data')
```

6.4 Download MNIST training Data

The MNIST is a dataset of handwritten digits, available from this page http://yann.lecun.com/exdb/mnist/. It has a training set of 60,000 examples, and a test set of 10,000 examples. In this task we will use images for digits 3 and 7.

[31]:
```
trainset = torchvision.datasets.MNIST(root = data_folder,
                                      train = True,
                                      download = True,
                                      )
```

6.5 Subset of images

trainset.data has images and trainset.targets has the labels. Below we will create a Boolean mask for digits 3 and 7. We will use this mask to extract subset of images.

[32]:
```
# get the mask
idx3 = trainset.targets==3
idx7 = trainset.targets==7
```

[33]:
```
idx3 # This take the value of True wherever the label was 3 and False otherwise
```

[33]:
```
tensor([False, False, False,  …, False, False, False])
```

[34]:
```
threes = trainset.data[idx3] # use Boolean mask to extract images for digit␣
↪three
sevens = trainset.data[idx7] # use Boolean mask to extract images for digit␣
↪seven
```

[35]:
```
threes.shape, sevens.shape
```

[35]:
```
(torch.Size([6131, 28, 28]), torch.Size([6265, 28, 28]))
```

[36]:
```
type(threes[0])
```

[36]:
```
torch.Tensor
```

[37]:
```
# get a sample image
sample_img_3 = threes[0]
```

[38]:
```
sample_img_3.shape
```

[38]:
```
torch.Size([28, 28])
```

6.5.1 Task1

Reshape the image so that it has following shape: 28, 28, 1. Then convert the tensor to numpy array.

[39]:
```
sample_img_3_numpy =  sample_img_3.unsqueeze(2).numpy()
```

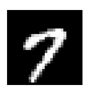
[40]:
```
# let us look at the image
# the above steps were needed because the function cv2_imshow needs a three dimenional numpy array of the
# shape H x W  X C (height, width, number of channels). The number of channels for black and white images is 1.
cv2_imshow(sample_img_3_numpy)
```

6.5.2 Task2

Repeat the above steps to show the first image for digit 7. - Get the first image. - Reshape the image to 28, 28, 1 - Convert the tensor to numpy array

[41]:
```
sample_img_7_numpy = sevens[0].unsqueeze(2).numpy()
```

[42]:
```
cv2_imshow(sample_img_7_numpy)
```



6.6 Visualizing images as pixel values

- In a computer, images are stored in the form of matrices. The numbers in the matrix are called "pixel values." - These pixel values show how bright each pixel is. - 0 means "black," and 255 means "white." - The matrix of numbers is called the channel, and there is only one channel in a grayscale image. - For color inages , we have three channels(RGB - Red, Green Blue). That is we need three matrices to store color images.

[43]:
```
df = pd.DataFrame(sample_img_3[5:24, 7:22].numpy())
df.style.set_properties(**{'font-size':'8pt'})
```

[43]:
```
<pandas.io.formats.style.Styler at 0x7b32c9e63730>
```

- In the matrix below we have flipped color just for illustartion (We are using 0 for white instead of black).

[44]:
```
df.style.set_properties(**{'font-size':'8pt'}).background_gradient('Greys')
```

[44]: <pandas.io.formats.style.Styler at 0x7b32c9e63c70>

6.6.1 Task3

Convert threes and sevens generated earlier to float and and take the mean along dimension 0.

Hint(You can use tensor.float() and then take the mean along dimesnion 0.

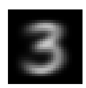
Here we are generating an average image by taking the average of each pixel across images.

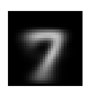
[45]:
```
mean_threes = threes.mean(dim=0, dtype=torch.float32)
mean_sevens = sevens.mean(dim=0, dtype=torch.float32)
```

[46]:
```
mean_threes.shape
mean_sevens.shape
```

[46]: torch.Size([28, 28])

[47]:
```
cv2_imshow(mean_threes.unsqueeze(dim=2).numpy())
# print()
cv2_imshow(mean_sevens.unsqueeze(dim=2).numpy())
```





6.7 Prediction for validation images

We will calculate the distance of each validation image from the mean_threes and mean_sevens. If the validation image is closest to mean_threes then we will predict 1 else we will predict 0.

6.8 Get Valid Dataset

[48]:
```
validset = torchvision.datasets.MNIST(root = data_folder,
                                      train = False,
                                      download = True,
                                      )
```

[49]:
```
# get the mask
idx3 = validset.targets==3
idx7 = validset.targets==7
```

[50]:
```
validset.data.shape
```

[50]: torch.Size([10000, 28, 28])

[51]:
```
# get images and labels using the mask
valid_data_3_7 = validset.data[idx3+idx7]
valid_targets_3_7 = validset.targets[idx3 + idx7]
```

[52]:
```
# check the shape of the inputs (images)
# we have 2038 images with size 28 x 28.
valid_data_3_7.shape
```

[52]: torch.Size([2038, 28, 28])

[53]:
```
# lables for valid dataset
valid_targets_3_7
```

[53]: tensor([7, 7, 3, …, 3, 7, 3])

[54]:
```
# change the lable to 1 where the label was 3
valid_targets_3_7[valid_targets_3_7==3] = 1
```

[55]:
```
# change the lable to 0 where the label was 7
valid_targets_3_7[valid_targets_3_7==7] = 0
```

[56]:
```
valid_targets_3_7.unique()
```

[56]: tensor([0, 1])

[57]:
```
valid_targets_3_7
```

[57]: tensor([0, 0, 1, …, 1, 0, 1])

[58]:
```
valid_data_3_7.shape
```

[58]: torch.Size([2038, 28, 28])

6.9 Distance between images

6.9.1 Task4

Write a function to calculate the distance between two images a and b. - Calculate the difference between corresponding pixels of two images. - Calculate square of differences. - Take the mean (take the mean across last two dimensions (-1, -2) of the square of the differences and then take the square root.

[59]:
```
def dist_images(a, b):
    diff = a.float() - b.float()
    squared_diff = diff ** 2
    distance = torch.sqrt(squared_diff.mean((-1, -2)))
    return distance
```

[60]:
```
# calculate the distance of the each image in validation set from mean_threes
# bacause of the  broadcasting we are able to use the above function which was␣
↪written to
# calculate the distnace between two images
valid_3_dist = dist_images(valid_data_3_7, mean_threes)
```

[61]:
```
# calculate the distance of the each image in validation set from mean_sevens
valid_7_dist = dist_images(valid_data_3_7, mean_sevens)
```

[62]:
```
# if you have done everything right, you will observe following shapes.
valid_3_dist.shape, valid_7_dist.shape
```

[62]:
```
(torch.Size([2038]), torch.Size([2038]))
```

6.10 Prediction Using Distance

6.10.1 Task5

[63]:
```
# step 1 : generate the tensor filled with zeros that has the same shape as␣
↪valid_target_3_7
prediction = torch.zeros_like(valid_targets_3_7)
```

[64]:
```
# step 2: update the tensor prediction. Update the value to 1, when␣
↪valid_3_dist < valid_7_dist.
# Hint (Generate a Boolean mask using the condition valid_3_dist < valid_7_dist␣
↪and use the mask to update value to 1 )

mask = valid_3_dist < valid_7_dist
prediction[mask] = 1
```

[65]:
```
prediction
```

[65]:
```
tensor([0, 0, 1,  …, 1, 0, 1])
```

6.11 Accuracy

[66]:
```
correct = prediction == valid_targets_3_7
```

[67]:
```
correct
```

[67]:
```
tensor([True, True, True,  …, True, True, True])
```

[68]:
```
100 * correct.float().mean().item()
```

[68]:
```
96.6143250465393
```