Lab 1. PyTorch and ANNs

This lab is a warm up to get you used to the PyTorch programming environment used in the course, and also to help you review and renew your knowledge of Python and relevant Python libraries. The lab must be done individually. Please recall that the University of Toronto plagarism rules apply.

By the end of this lab, you should be able to:

1. Be able to perform basic PyTorch tensor operations.
2. Be able to load data into PyTorch
3. Be able to configure an Artificial Neural Network (ANN) using PyTorch
4. Be able to train ANNs using PyTorch
5. Be able to evaluate different ANN configuations

You will need to use numpy and PyTorch documentations for this assignment:

- https://docs.scipy.org/doc/numpy/reference/
- https://pytorch.org/docs/stable/torch.html

You can also reference Python API documentations freely.

What to submit

Submit a PDF file containing all your code, outputs, and write-up from parts 1-5. You can produce a PDF of your Google Colab file by going to `File -> Print` and then save as PDF. The Colab instructions has more information.

Do not submit any other files produced by your code.

Include a link to your colab file in your submission.

Please use Google Colab to complete this assignment. If you want to use Jupyter Notebook, please complete the assignment and upload your Jupyter Notebook file to Google Colab for submission.

Adjust the scaling to ensure that the text is not cutoff at the margins.

Colab Link

Submit make sure to include a link to your colab file here

Colab Link: https://colab.research.google.com/drive/17VloI-ljJxAEeX8Hldtvekm_6O2_OOk1?usp=sharing

Part 1. Python Basics [3 pt]

The purpose of this section is to get you used to the basics of Python, including working with functions, numbers, lists, and strings.

Note that we **will** be checking your code for clarity and efficiency.

If you have trouble with this part of the assignment, please review http://cs231n.github.io/python-numpy-tutorial/

Part (a) -- 1pt

Write a function `sum_of_cubes` that computes the sum of cubes up to `n`. If the input to `sum_of_cubes` invalid (e.g. negative or non-integer `n`), the function should print out `"Invalid input"` and return `-1`.

```
def sum_of_cubes(n):
    """Return the sum (1^3 + 2^3 + 3^3 + ... + n^3)

    Precondition: n > 0, type(n) == int

    >>> sum_of_cubes(3)
    36
    >>> sum_of_cubes(1)
    1
    """
    if type(n) is not int:
      print("Invalid input")
    if n <= 0:
      print("Invalid input")
      return -1
    result = 0
    for i in range(1, n+1):
      result += i**3
    return result

sum_of_cubes(5)
```

```
225
```

Part (b) -- 1pt

Write a function `word_lengths` that takes a sentence (string), computes the length of each word in that sentence, and returns the length of each word in a list. You can assume that words are always separated by a space character `" "`.

Hint: recall the `str.split` function in Python. If you arenot sure how this function works, try typing `help(str.split)` into a Python shell, or check out https://docs.python.org/3.6/library/stdtypes.html#str.split

```
help(str.split)
```

```
Help on method_descriptor:

split(self, /, sep=None, maxsplit=-1)
    Return a list of the substrings in the string, using sep as the separator string.

      sep
        The separator used to split the string.

        When set to None (the default value), will split on any whitespace
        character (including \\n \\r \\t \\f and spaces) and will discard
        empty strings from the result.
      maxsplit
        Maximum number of splits (starting from the left).
        -1 (the default value) means no limit.

    Note, str.split() is mainly useful for data that has been intentionally
    delimited.  With natural text that includes punctuation, consider using
    the regular expression module.
```

```
def word_lengths(sentence):
    """Return a list containing the length of each word in
    sentence.

    >>> word_lengths("welcome to APS360!")
    [7, 2, 7]
    >>> word_lengths("machine learning is so cool")
    [7, 8, 2, 2, 4]
    """
    sentence = sentence.strip()
    words = sentence.split(" ")
    result = []
    for w in words:
      result.append(len(w))
    return result

word_lengths("welcome to APS360!")
```

```
[7, 2, 7]
```

Part (c) -- 1pt

Write a function `all_same_length` that takes a sentence (string), and checks whether every word in the string is the same length. You should call the function `word_lengths` in the body of this new function.

```
def all_same_length(sentence):
    """Return True if every word in sentence has the same
    length, and False otherwise.

    >>> all_same_length("all same length")
    False
    >>> word_lengths("hello world")
    True
    """
    wl = word_lengths(sentence)
    if len(set(wl)) == 1:
      return True
    else:
      return False
all_same_length("hello world")
```

```
True
```

Part 2. NumPy Exercises [5 pt]

In this part of the assignment, you'll be manipulating arrays usign NumPy. Normally, we use the shorter name `np` to represent the package `numpy`.

```
import numpy as np
```

Part (a) -- 1pt

The below variables `matrix` and `vector` are numpy arrays. Explain what you think `<NumpyArray>.size` and `<NumpyArray>.shape` represent.

```
matrix = np.array([[1., 2., 3., 0.5],
                   [4., 5., 0., 0.],
                   [-1., -2., 1., 1.]])
vector = np.array([2., 0., 1., -2.])
```

```
matrix.size #Number of elements in the matrix
```

```
12
```

```
matrix.shape #the dimensions of the matrix, in this case 3*4
```

```
(3, 4)
```

```
vector.size #Number of elements in the vector
```

```
4
```

```
vector.shape #the dimensions of the vector, in this case 4 from the 1-D Array
```

```
(4,)
```

Part (b) -- 1pt

Perform matrix multiplication `output = matrix x vector` by using for loops to iterate through the columns and rows. Do not use any builtin NumPy functions. Cast your output into a NumPy array, if it isn't one already.

Hint: be mindful of the dimension of output

```
output = None
```

```
output = []
for r in matrix:
  sum = 0
  for c in range(len(r)):
    sum += vector[c]*r[c]
  output.append(sum)
output = np.array(output)
```

```
output
```

```
array([ 4.,  8., -3.])
```

Part (c) -- 1pt

Perform matrix multiplication `output2 = matrix x vector` by using the function `numpy.dot`.

We will never actually write code as in part(c), not only because `numpy.dot` is more concise and easier to read/write, but also performance-wise `numpy.dot` is much faster (it is written in C and highly optimized). In general, we will avoid for loops in our code.

```
output2 = None
```

```
output2 = np.dot(matrix,vector)
```

```
output2
```

```
array([ 4.,  8., -3.])
```

Part (d) -- 1pt

As a way to test for consistency, show that the two outputs match.

```
if np.array_equal(output,output2):
  print("The outputs match")
else:
  print("The outputs did not match")
```

```
The outputs match
```

Part (e) -- 1pt

Show that using `np.dot` is faster than using your code from part (c).

You may find the below code snippit helpful:

```
import time

# record the time before running code
start_time = time.time()

# place code to run here
for i in range(10000):
  output2 = np.dot(matrix,vector)

# record the time after the code is run
end_time = time.time()

# compute the difference
diff = end_time - start_time
diff
```

```
0.0349733829498291
```

```
import time

# record the time before running code
start_time = time.time()

# place code to run here
for i in range(10000):
  output = []
  for r in matrix:
    sum = 0
    for c in range(len(r)):
      sum += vector[c]*r[c]
    output.append(sum)
  output = np.array(output)
# record the time after the code is run
end_time = time.time()

# compute the difference
```

```
diff = end_time - start_time
diff
```

```
0.20103788375854492
```

Part 3. Images [6 pt]

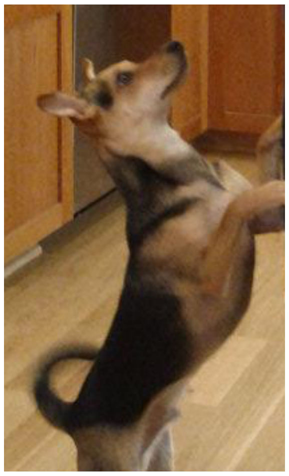
A picture or image can be represented as a NumPy array of “pixels”, with dimensions H × W × C, where H is the height of the image, W is the width of the image, and C is the number of colour channels. Typically we will use an image with channels that give the the Red, Green, and Blue “level” of each pixel, which is referred to with the short form RGB.

You will write Python code to load an image, and perform several array manipulations to the image and visualize their effects.

```
import matplotlib.pyplot as plt
```

Part (a) -- 1 pt

This is a photograph of a dog whose name is Mochi.



Load the image from its url (https://drive.google.com/uc?export=view&id=1oaLVR2hr1_qzpKQ47i9rVUIklwbDcews) into the variable `img` using the `plt.imread` function.

Hint: You can enter the URL directly into the `plt.imread` function as a Python string.

```
import gdown

gdown.download("https://drive.google.com/uc?export=view&id=1oaLVR2hr1_qzpKQ47i9rVUIklwbDcews", "image.jpg", quiet=False)
img = plt.imread("image.jpg")
```

```
Downloading...
From: https://drive.google.com/uc?export=view&id=1oaLVR2hr1_qzpKQ47i9rVUIklwbDcews
To: /content/image.jpg
100%|██████████| 109k/109k [00:00<00:00, 45.4MB/s]
```

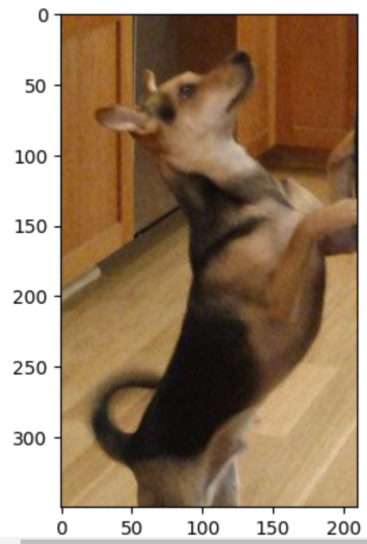
Part (b) -- 1pt

Use the function `plt.imshow` to visualize `img`.

This function will also show the coordinate system used to identify pixels. The origin is at the top left corner, and the first dimension indicates the Y (row) direction, and the second dimension indicates the X (column) dimension.

```
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7b0ccc263970>
```

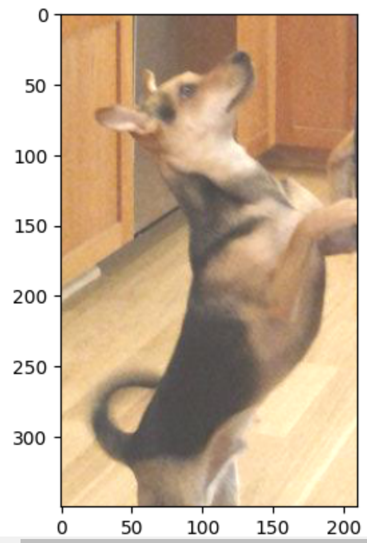


Part (c) -- 2pt

Modify the image by adding a constant value of 0.25 to each pixel in the `img` and store the result in the variable `img_add`. Note that, since the range for the pixels needs to be between [0, 1], you will also need to clip img_add to be in the range [0, 1] using `numpy.clip`. Clipping sets any value that is outside of the desired range to the closest endpoint. Display the image using `plt.imshow`.

```
img_add = None
img_add = img + 0.25
img_add = np.clip(img_add, 0, 1)
plt.imshow(img_add)
```

```
<matplotlib.image.AxesImage at 0x7b0ccc14de70>
```



Part (d) -- 2pt

Crop the **original** image (`img` variable) to a 130 x 150 image including Mochi's face. Discard the alpha colour channel (i.e. resulting `img_cropped` should **only have RGB channels**)

Display the image.

```
img.shape
```

```
(350, 210, 4)
```

```
img_cropped = img[:,:,:3]
img_cropped.shape
```

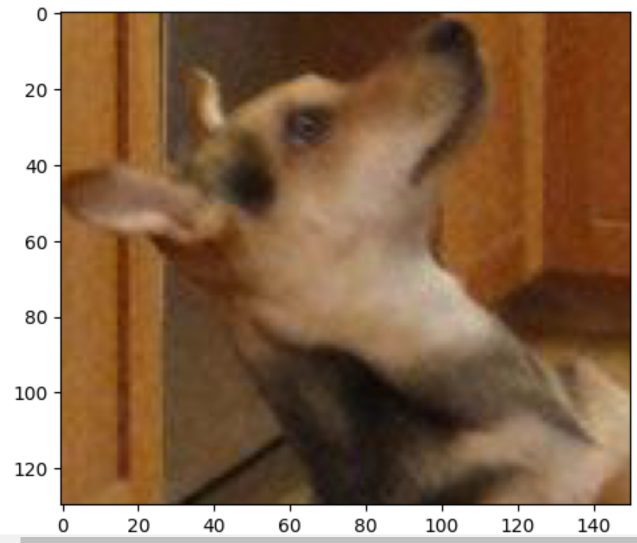
```
(350, 210, 3)
```

```
img_cropped = img_cropped[25:155, 25:175, :]
img_cropped.shape
```

```
(130, 150, 3)
```

```
plt.imshow(img_cropped)
```

```
<matplotlib.image.AxesImage at 0x7b0ccf31d030>
```



Part 4. Basics of PyTorch [6 pt]

PyTorch is a Python-based neural networks package. Along with tensorflow, PyTorch is currently one of the most popular machine learning libraries.

PyTorch, at its core, is similar to Numpy in a sense that they both try to make it easier to write codes for scientific computing achieve improved performance over vanilla Python by leveraging highly optimized C back-end. However, compare to Numpy, PyTorch offers much better GPU support and provides many high-level features for machine learning. Technically, Numpy can be used to perform almost every thing PyTorch does. However, Numpy would be a lot slower than PyTorch, especially with CUDA GPU, and it would take more effort to write machine learning related code compared to using PyTorch.

```
import torch
```

Part (a) -- 1 pt

Use the function `torch.from_numpy` to convert the numpy array `img_cropped` into a PyTorch tensor. Save the result in a variable called `img_torch`.

```
img_torch = None
```

```
img_torch = torch.from_numpy(img_cropped)
```

Part (b) -- 1pt

Use the method `<Tensor>.shape` to find the shape (dimension and size) of `img_torch`.

```
img_torch.shape
```

```
torch.Size([130, 150, 3])
```

Part (c) -- 1pt

How many floating-point numbers are stored in the tensor `img_torch`?

```
#130 * 150 * 3 = 58,500
```

Part (d) -- 1 pt

What does the code `img_torch.transpose(0,2)` do? What does the expression return? Is the original variable `img_torch` updated? Explain.

```
img_torch
```

```
tensor([[[0.6510, 0.4510, 0.2353],
         [0.6588, 0.4588, 0.2431],
         [0.6510, 0.4510, 0.2275],
         ...,
         [0.4980, 0.2431, 0.0784],
         [0.4941, 0.2392, 0.0745],
         [0.4941, 0.2471, 0.0784]],

        [[0.6588, 0.4588, 0.2510],
         [0.6588, 0.4588, 0.2431],
         [0.6510, 0.4510, 0.2353],
         ...,
         [0.4941, 0.2392, 0.0745],
         [0.4863, 0.2314, 0.0667],
         [0.4980, 0.2510, 0.0824]],

        [[0.6627, 0.4627, 0.2549],
         [0.6627, 0.4627, 0.2549],
         [0.6510, 0.4510, 0.2353],
         ...,
         [0.5176, 0.2627, 0.0980],
         [0.4941, 0.2392, 0.0745],
         [0.4902, 0.2431, 0.0745]],

        ...,

        [[0.6196, 0.4118, 0.2157],
         [0.6118, 0.4039, 0.2078],
         [0.5922, 0.3882, 0.1922],
         ...,
         [0.5373, 0.3647, 0.1882],
         [0.5529, 0.3765, 0.2118],
         [0.5686, 0.3922, 0.2314]],

        [[0.6157, 0.4118, 0.2235],
         [0.6000, 0.3961, 0.2078],
         [0.5843, 0.3804, 0.1922],
         ...,
         [0.5294, 0.3569, 0.1804],
         [0.5490, 0.3765, 0.2000],
         [0.5647, 0.3961, 0.2275]],

        [[0.6078, 0.4039, 0.2196],
         [0.6039, 0.4000, 0.2157],
         [0.5961, 0.3922, 0.2039],
         ...,
         [0.5333, 0.3608, 0.1765],
         [0.5373, 0.3686, 0.1922],
         [0.5529, 0.3843, 0.2157]]])
```

```
img_torch.transpose(0,2)
```

```
tensor([[[0.6510, 0.6588, 0.6627,  ..., 0.6196, 0.6157, 0.6078],
         [0.6588, 0.6588, 0.6627,  ..., 0.6118, 0.6000, 0.6039],
         [0.6510, 0.6510, 0.6510,  ..., 0.5922, 0.5843, 0.5961],
         ...,
         [0.4980, 0.4941, 0.5176,  ..., 0.5373, 0.5294, 0.5333],
         [0.4941, 0.4863, 0.4941,  ..., 0.5529, 0.5490, 0.5373],
         [0.4941, 0.4980, 0.4902,  ..., 0.5686, 0.5647, 0.5529]],

        [[0.4510, 0.4588, 0.4627,  ..., 0.4118, 0.4118, 0.4039],
         [0.4588, 0.4588, 0.4627,  ..., 0.4039, 0.3961, 0.4000],
         [0.4510, 0.4510, 0.4510,  ..., 0.3882, 0.3804, 0.3922],
         ...,
         [0.2431, 0.2392, 0.2627,  ..., 0.3647, 0.3569, 0.3608],
         [0.2392, 0.2314, 0.2392,  ..., 0.3765, 0.3765, 0.3686],
         [0.2471, 0.2510, 0.2431,  ..., 0.3922, 0.3961, 0.3843]],

        [[0.2353, 0.2510, 0.2549,  ..., 0.2157, 0.2235, 0.2196],
         [0.2431, 0.2431, 0.2549,  ..., 0.2078, 0.2078, 0.2157],
         [0.2275, 0.2353, 0.2353,  ..., 0.1922, 0.1922, 0.2039],
         ...,
         [0.0784, 0.0745, 0.0980,  ..., 0.1882, 0.1804, 0.1765],
         [0.0745, 0.0667, 0.0745,  ..., 0.2118, 0.2000, 0.1922],
         [0.0784, 0.0824, 0.0745,  ..., 0.2314, 0.2275, 0.2157]]])
```

```
img_torch
```

```
tensor([[[0.6510, 0.4510, 0.2353],
         [0.6588, 0.4588, 0.2431],
         [0.6510, 0.4510, 0.2275],
         ...,
         [0.4980, 0.2431, 0.0784],
         [0.4941, 0.2392, 0.0745],
         [0.4941, 0.2471, 0.0784]],

        [[0.6588, 0.4588, 0.2510],
         [0.6588, 0.4588, 0.2431],
         [0.6510, 0.4510, 0.2353],
         ...,
         [0.4941, 0.2392, 0.0745],
         [0.4863, 0.2314, 0.0667],
         [0.4980, 0.2510, 0.0824]],

        [[0.6627, 0.4627, 0.2549],
         [0.6627, 0.4627, 0.2549],
         [0.6510, 0.4510, 0.2353],
         ...,
         [0.5176, 0.2627, 0.0980],
         [0.4941, 0.2392, 0.0745],
         [0.4902, 0.2431, 0.0745]],

        ...,

        [[0.6196, 0.4118, 0.2157],
         [0.6118, 0.4039, 0.2078],
         [0.5922, 0.3882, 0.1922],
         ...,
         [0.5373, 0.3647, 0.1882],
         [0.5529, 0.3765, 0.2118],
         [0.5686, 0.3922, 0.2314]],

        [[0.6157, 0.4118, 0.2235],
         [0.6000, 0.3961, 0.2078],
         [0.5843, 0.3804, 0.1922],
         ...,
         [0.5294, 0.3569, 0.1804],
         [0.5490, 0.3765, 0.2000],
         [0.5647, 0.3961, 0.2275]],

        [[0.6078, 0.4039, 0.2196],
         [0.6039, 0.4000, 0.2157],
         [0.5961, 0.3922, 0.2039],
         ...,
         [0.5333, 0.3608, 0.1765],
         [0.5373, 0.3686, 0.1922],
         [0.5529, 0.3843, 0.2157]]])
```

```
print(img_torch.shape)
print(img_torch.transpose(0,2).shape)
```

```
torch.Size([130, 150, 3])
torch.Size([3, 150, 130])
```

The code img_torch.transpose(0,2) swaps the first dimension with the third dimension of img_torch. The first dimension of the transposed img_torch will now be representing R,G,B, and the second and third dimension will now be Width, height. The original img_torch is not updated, unless you run img_torch = img_torch.transpose(0,2)

Part (e) -- 1 pt

What does the code `img_torch.unsqueeze(0)` do? What does the expression return? Is the original variable `img_torch` updated? Explain.

```
img_torch.unsqueeze(0).shape
```

```
torch.Size([1, 130, 150, 3])
```

```
img_torch.shape
```

```
torch.Size([130, 150, 3])
```

It turns the dimensions of img_torch from 3D to 4D with the additional dimension in front. So, it just returns img_torch with additional bracket at the start and at the end. the variable img_torch will not be updated, unless you run img_torch = img_torch.unsqueeze(0)

Part (f) -- 1 pt

Find the maximum value of `img_torch` along each colour channel? Your output should be a one-dimensional PyTorch tensor with exactly three values.

Hint: lookup the function `torch.max`.

```
max_row = torch.max(img_torch,0).values
max_row.shape
```

```
torch.Size([150, 3])
```

```
max_img_torch = torch.max(max_row,0).values
max_img_torch
```

```
tensor([0.8941, 0.7882, 0.6745])
```

Part 5. Training an ANN [10 pt]

The sample code provided below is a 2-layer ANN trained on the MNIST dataset to identify digits less than 3 or greater than and equal to 3. Modify the code by changing any of the following and observe how the accuracy and error are affected:

- number of training iterations
- number of hidden units
- numbers of layers
- types of activation functions
- learning rate

Please select at least three different options from the list above. For each option, please select two to three different parameters and provide a table.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
import matplotlib.pyplot as plt # for plotting
import torch.optim as optim

torch.manual_seed(1) # set the random seed

# define a 2-layer artificial neural network
class Pigeon(nn.Module):
    def __init__(self):
        super(Pigeon, self).__init__()
        self.layer1 = nn.Linear(28 * 28, 30)
        self.layer2 = nn.Linear(30, 1)
    def forward(self, img):
        flattened = img.view(-1, 28 * 28)
        activation1 = self.layer1(flattened)
        activation1 = F.relu(activation1)
        activation2 = self.layer2(activation1)
        return activation2

pigeon = Pigeon()

# load the data
mnist_data = datasets.MNIST('data', train=True, download=True)
mnist_data = list(mnist_data)
mnist_train = mnist_data[:1000]
mnist_val   = mnist_data[1000:2000]
img_to_tensor = transforms.ToTensor()


# simplified training code to train `pigeon` on the "small digit recognition" task
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.SGD(pigeon.parameters(), lr=0.005, momentum=0.9)

for (image, label) in mnist_train:
    # actual ground truth: is the digit less than 3?
    actual = torch.tensor(label < 3).reshape([1,1]).type(torch.FloatTensor)
    # pigeon prediction
    out = pigeon(img_to_tensor(image)) # step 1-2
    # update the parameters based on the loss
    loss = criterion(out, actual)      # step 3
    loss.backward()                    # step 4 (compute the updates for each parameter)
    optimizer.step()                   # step 4 (make the updates for each parameter)
    optimizer.zero_grad()              # a clean up step for PyTorch

# computing the error and accuracy on the training set
error = 0
for (image, label) in mnist_train:
    prob = torch.sigmoid(pigeon(img_to_tensor(image)))
    if (prob < 0.5 and label < 3) or (prob >= 0.5 and label >= 3):
        error += 1
print("Training Error Rate:", error/len(mnist_train))
print("Training Accuracy:", 1 - error/len(mnist_train))


# computing the error and accuracy on a test set
error = 0
for (image, label) in mnist_val:
    prob = torch.sigmoid(pigeon(img_to_tensor(image)))
    if (prob < 0.5 and label < 3) or (prob >= 0.5 and label >= 3):
        error += 1
print("Test Error Rate:", error/len(mnist_val))
print("Test Accuracy:", 1 - error/len(mnist_val))
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz to data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 32464082.22it/s]
Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 1617255.80it/s]
Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz to data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:00<00:00, 2293014.22it/s]
Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz to data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4542/4542 [00:00<00:00, 2347569.78it/s]
Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw

Training Error Rate: 0.036
Training Accuracy: 0.964
Test Error Rate: 0.079
Test Accuracy: 0.921
```

I choose learning rate, number of layers, number of training iterations.

Grid:

Learning rate: [0.001,0.01,0.1]

Number of Layers: [1,3,5]

Number of Training Iterations: [1,2,3]

I will put my result in a 3-D Matrix with respect to the grid defined above. For example, matrix[0][0][0] will represents 0.001 learning rate, 1 layers, 1 iterations; matrix[0][1][2] will represents 0.001 learning rate, 3 layers, 3 iterations.

```
from tqdm import tqdm


def NN_testing(learning_rate,num_layers,num_epochs):
  torch.manual_seed(1) # set the random seed
  class Pigeon_Layers(nn.Module):
      def __init__(self, num_layers):
          #num_layers is the number of hidden layers
          super(Pigeon_Layers, self).__init__()
          self.layers = nn.ModuleList()
          self.layers.append(nn.Linear(28*28,30))
          for i in range(1,num_layers):
            self.layers.append(nn.Linear(30,30))
          self.output_layer = nn.Linear(30,1) #output layer

      def forward(self, img):
          flattened = img.view(-1, 28 * 28)
          activation = flattened
          for layer in self.layers:
            activation = layer(activation)
            activation = F.relu(activation)
          activation = self.output_layer(activation)

          return activation
  pigeon = Pigeon_Layers(num_layers)

  # simplified training code to train `pigeon` on the "small digit recognition" task
  criterion = nn.BCEWithLogitsLoss()
  optimizer = optim.SGD(pigeon.parameters(), lr=learning_rate, momentum=0.9)
  for i in range(num_epochs):
    for (image, label) in mnist_train:
        # actual ground truth: is the digit less than 3?
        actual = torch.tensor(label < 3).reshape([1,1]).type(torch.FloatTensor)
        # pigeon prediction
        out = pigeon(img_to_tensor(image)) # step 1-2
        # update the parameters based on the loss
        loss = criterion(out, actual)      # step 3
        loss.backward()                    # step 4 (compute the updates for each parameter)
        optimizer.step()                   # step 4 (make the updates for each parameter)
        optimizer.zero_grad()              # a clean up step for PyTorch

  # computing the error and accuracy on the training set
  error = 0
  for (image, label) in mnist_train:
      prob = torch.sigmoid(pigeon(img_to_tensor(image)))
      if (prob < 0.5 and label < 3) or (prob >= 0.5 and label >= 3):
          error += 1
  train_accuracy =  1 - error/len(mnist_train)


  # computing the error and accuracy on a test set
  error = 0
  for (image, label) in mnist_val:
      prob = torch.sigmoid(pigeon(img_to_tensor(image)))
      if (prob < 0.5 and label < 3) or (prob >= 0.5 and label >= 3):
          error += 1
  test_accuracy = 1 - error/len(mnist_val)
  return (train_accuracy,test_accuracy)
```

```
lr = [0.001, 0.01, 0.1]
```

```
no_of_layers = [1,3,5]
epochs = [1,2,3]
result_train = np.full((3, 3, 3), np.nan)
result_test = np.full((3, 3, 3), np.nan)
for i in tqdm(range(len(lr)),desc="Learning rate"):
  for j in tqdm(range(len(no_of_layers)),desc="Number of Layers"):
    for k in range(len(epochs)):
      train_accuracy, test_accuracy = NN_testing(lr[i],no_of_layers[j],epochs[k])
      result_train[i][j][k] = train_accuracy
      result_test[i][j][k] = test_accuracy
```

```
Learning rate:   0%|          | 0/3 [00:00<?, ?it/s]
Number of Layers:   0%|          | 0/3 [00:00<?, ?it/s]
Number of Layers:  33%|███▍      | 1/3 [00:07<00:15,  7.67s/it]
Number of Layers:  67%|██████▋   | 2/3 [00:17<00:08,  8.95s/it]
Number of Layers: 100%|██████████| 3/3 [00:29<00:00,  9.95s/it]
Learning rate:  33%|███▎      | 1/3 [00:29<00:59, 29.86s/it]
Number of Layers:   0%|          | 0/3 [00:00<?, ?it/s]
Number of Layers:  33%|███▍      | 1/3 [00:07<00:14,  7.45s/it]
Number of Layers:  67%|██████▋   | 2/3 [00:17<00:09,  9.17s/it]
Number of Layers: 100%|██████████| 3/3 [00:30<00:00, 10.14s/it]
Learning rate:  67%|██████▋   | 2/3 [01:00<00:30, 30.20s/it]
Number of Layers:   0%|          | 0/3 [00:00<?, ?it/s]
Number of Layers:  33%|███▍      | 1/3 [00:08<00:16,  8.13s/it]
Number of Layers:  67%|██████▋   | 2/3 [00:18<00:09,  9.65s/it]
Number of Layers: 100%|██████████| 3/3 [00:32<00:00, 10.92s/it]
Learning rate: 100%|██████████| 3/3 [01:33<00:00, 31.02s/it]
```

```
result_train
```

```
array([[[0.922, 0.96 , 0.968],
        [0.872, 0.955, 0.964],
        [0.688, 0.688, 0.688]],

       [[0.961, 0.975, 0.951],
        [0.951, 0.956, 0.962],
        [0.899, 0.688, 0.688]],

       [[0.688, 0.688, 0.688],
        [0.688, 0.688, 0.688],
        [0.688, 0.688, 0.688]]])
```

```
result_test
```

```
array([[[0.887, 0.906, 0.913],
        [0.83 , 0.896, 0.911],
        [0.703, 0.703, 0.703]],

       [[0.918, 0.931, 0.899],
        [0.901, 0.907, 0.911],
        [0.894, 0.703, 0.703]],

       [[0.703, 0.703, 0.703],
        [0.703, 0.703, 0.703],
        [0.703, 0.703, 0.703]]])
```

Part (a) -- 3 pt

Comment on which of the above changes resulted in the best accuracy on training data? What accuracy were you able to achieve?

```
flat_index = np.argmax(result_train)
max_index = np.unravel_index(flat_index,result_train.shape)
max = np.max(result_train)
print(f"Max Accuracy: {max}, using {max_index}")
```

```
Max Accuracy: 0.975, using (1, 0, 1)
```

Learning Rate = 0.01, Number of hidden layers = 1, Number of training iterations = 2

Part (b) -- 3 pt

Comment on which of the above changes resulted in the best accuracy on testing data? What accuracy were you able to achieve?

```
flat_index = np.argmax(result_test)
max_index = np.unravel_index(flat_index,result_test.shape)
max = np.max(result_test)
print(f"Max Accuracy: {max}, using {max_index}")
```

```
Max Accuracy: 0.931, using (1, 0, 1)
```

Learning Rate = 0.01, Number of hidden layers = 1, Number of training iterations = 2

Part (c) -- 4 pt

Which model hyperparameters should you use, the ones from (a) or (b)?

(b) because high accuracy in training data can be a result from overfitting which means that the model does very good on the training data but cannot generalize well, leading to bad performance when testing.