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://drive.google.com/file/d/1-Nqblp2Hc4zo5zXXNSf_DtFdFn9lrBSu/view?usp=share_link

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`.

In [34]:
```
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 n <= 0 or type(n) != int:
      print("Invaid input!")
      return -1

    else:
      sum = 0
      while n != 0:
        sum += n**3
        n -= 1
      return sum

# Test:
print(sum_of_cubes(3))
print(sum_of_cubes(1))
```

```
36
1
```

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

In [35]:
```
help(str.split)
```

```
Help on method_descriptor:

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

    sep
      The delimiter according which to split the string.
      None (the default value) means split according to any whitespace,
      and discard empty strings from the result.
    maxsplit
      Maximum number of splits to do.
      -1 (the default value) means no limit.
```

In [36]:
```
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]
    """
    words = sentence.split()
    lengths = [len(word) for word in words]
    return lengths

# Test:
print(word_lengths("welcome to APS360!"))
print(word_lengths("machine learning is so cool"))
```

```
[7, 2, 7]
[7, 8, 2, 2, 4]
```

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.

In [37]:
```
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
    """
    lengths = {x for x in word_lengths(sentence)}
    return len(lengths) == 1

# Tests:
print(all_same_length("all same length"))
print(all_same_length("hello world"))
```

```
False
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`.

In [38]:
```
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.

In [39]:
```
matrix = np.array([[1., 2., 3., 0.5],
                   [4., 5., 0., 0.],
                   [-1., -2., 1., 1.]])
vector = np.array([2., 0., 1., -2.])
```

In [40]:
```
matrix.size
```

Out[40]:
```
12
```

In [41]:
```
matrix.shape
```

Out[41]:
```
(3, 4)
```

In [42]:
```
vector.size
```

Out[42]:
```
4
```

In [43]:
```
vector.shape
```
Out[43]:
```
(4,)
```

ANSWER:

< NumpyArray >.size gives us the number of terms in the Array

and < NumpyArray >.shape gives us the dimensions of the Array

-> (# of terms or lists = rows, # of terms in each nested list = columns)

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

In [44]:
```
output = None
```
In [45]:
```
output = []
for row in matrix:
  sum = 0
  for i in range(len(row)):
    sum += row[i]*vector[i]
  output.append(sum)

output = np.array(output)
print(output)
```
```
[ 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.

In [46]:
```
output2 = None
```
In [47]:
```
output2 = np.dot(matrix,vector)
print(output2)
```
```
[ 4.  8. -3.]
```

Part (d) -- 1pt

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

In [48]:
```
print(output == output2)
```
```
[ True  True  True]
```

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:

In [49]:
```
import time

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

# place code to run here
output = []
for row in matrix:
  sum = 0
  for i in range(len(row)):
    sum += row[i]*vector[i]
  output.append(sum)

output = np.array(output)
print(output)

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

# compute the difference
diff = end_time - start_time
print("Time with for loops:", diff)
```
```
[ 4.  8. -3.]
Time with for loops: 0.004207611083984375
```
In [50]:
```
# record the time before running code
start_time = time.time()

# place code to run here
output2 = np.dot(matrix,vector)
print(output2)

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

# compute the difference
diff = end_time - start_time
print("Time with for loops:", diff)
```
```
[ 4.  8. -3.]
Time with for loops: 0.0016837120056152344
```

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.

In [51]:
```
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.

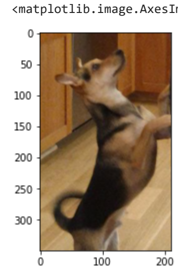
In [52]:
```
img = plt.imread("https://drive.google.com/uc?export=view&id=1oaLVR2hr1_qzpKQ47i9rV
```

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.

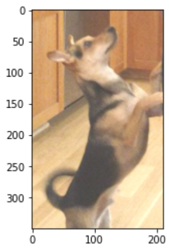
In [53]:
```
plt.imshow(img)
```
Out[53]:
```
<matplotlib.image.AxesImage at 0x7f7b00e9cf40>
```



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`.

In [54]:
```
img_add = np.clip(img + 0.25,0,1)
plt.imshow(img_add)
```
Out[54]:
```
<matplotlib.image.AxesImage at 0x7f7af5381bb0>
```



Part (d) -- 2pt

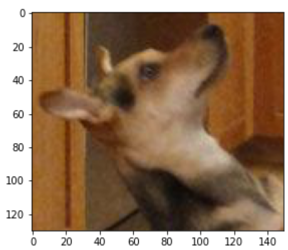
Crop the **original** image (`img` variable) to a 130 x 150 image including Mochi's face. Discard the alpha colour channel (i.e. resulting `img_cropped` should **only have RGB channels**)

Display the image.

In [55]:
```
img_cropped = img[20:150, 20:170, :3] # Third term contains the channels, only the
plt.imshow(img_cropped)

# Citation: Stackoverflow: https://stackoverflow.com/questions/35902302/discarding-
```

Out[55]:
```
<matplotlib.image.AxesImage at 0x7f7af359eb50>
```



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.

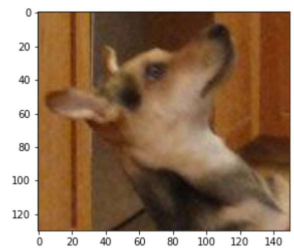
In [56]:
```
import torch
```

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`.

In [64]:
```
img_torch = torch.from_numpy(img_cropped)
plt.imshow(img_torch)
```

Out[64]:
```
<matplotlib.image.AxesImage at 0x7f7affb965e0>
```



Part (b) -- 1pt

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

In [58]:
```
print(img_torch.shape)
```

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

Part (c) -- 1pt

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

In [59]:
```
print("Total of Floating points = 130 * 150 * 3 =", 130*150*3) # = 58500
```

```
Total of Floating points = 130 * 150 * 3 = 58500
```

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.

In [60]:
```
print(img_torch.transpose(0,2))
print(img_torch.transpose(0,2).shape)
print(img_torch)
print(img_torch.shape)

# The transpose function switched the 1st and 3rd dimension of img_torch (shown by
# It doesn't change the original variable img_torch (can see the original dimension
```

```
tensor([[[0.6353, 0.6392, 0.6392,  ..., 0.5961, 0.6078, 0.6118],
         [0.6431, 0.6392, 0.6314,  ..., 0.5804, 0.6000, 0.6000],
         [0.6510, 0.6353, 0.6235,  ..., 0.5961, 0.6078, 0.6039],
         ...,
         [0.4627, 0.4784, 0.4941,  ..., 0.7529, 0.7490, 0.6667],
         [0.4784, 0.5098, 0.5137,  ..., 0.7333, 0.6667, 0.6118],
         [0.5059, 0.5176, 0.5098,  ..., 0.7059, 0.6314, 0.5882]],

        [[0.4353, 0.4392, 0.4392,  ..., 0.3765, 0.3882, 0.3922],
         [0.4431, 0.4353, 0.4275,  ..., 0.3608, 0.3804, 0.3804],
         [0.4510, 0.4314, 0.4196,  ..., 0.3765, 0.3882, 0.3843],
         ...,
         [0.2157, 0.2314, 0.2471,  ..., 0.6118, 0.6000, 0.5176],
         [0.2235, 0.2549, 0.2588,  ..., 0.5647, 0.4941, 0.4353],
         [0.2510, 0.2627, 0.2549,  ..., 0.5373, 0.4588, 0.4118]],

        [[0.2275, 0.2314, 0.2314,  ..., 0.1765, 0.1882, 0.1922],
         [0.2353, 0.2392, 0.2314,  ..., 0.1608, 0.1804, 0.1804],
         [0.2431, 0.2353, 0.2235,  ..., 0.1843, 0.1961, 0.1922],
         ...,
         [0.0471, 0.0627, 0.0784,  ..., 0.5255, 0.5176, 0.4353],
         [0.0667, 0.0980, 0.1020,  ..., 0.4275, 0.3412, 0.2745],
         [0.0941, 0.1059, 0.0980,  ..., 0.4000, 0.3059, 0.2510]]])
torch.Size([3, 150, 130])
```

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.

In [61]:
```
#help(torch.unsqueeze)
print(img_torch.unsqueeze(0))
print(img_torch.unsqueeze(0).shape)

# The unsqeeze function inputs a dimension of size one in the position specified in
# We can see an additional one in the 0th position in the Torch.Size
# This function does not alter the original variable as it creates a new matrix whi

# Citation: Pytorch Documentation
```

```
tensor([[[[0.6353, 0.4353, 0.2275],
          [0.6431, 0.4431, 0.2353],
          [0.6510, 0.4510, 0.2431],
          ...,
          [0.6667, 0.5176, 0.4353],
          [0.6118, 0.4353, 0.2745],
          [0.5882, 0.4118, 0.2510]]]])
torch.Size([1, 130, 150, 3])
```

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`.

In [62]:
```
#help(torch.max)
max_terms = torch.max(torch.max(img_torch,0).values,0).values
print(max_terms)

# First getting the max terms over the first dimension,
# and then the second dimension leaving use with the third dimension of size 3
# each with values of the maximum term in each channel from all the pizels.

# Citation: Pytorch Documentation
```

```
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.

In [68]:
```
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
```

```python
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)

# Num of epoch
epoch = 2

for i in range(epoch):
  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 par
      optimizer.step()                   # step 4 (make the updates for each parame
      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))
```

```
Training Error Rate: 0.016
Training Accuracy: 0.984
Test Error Rate: 0.057
Test Accuracy: 0.943
```

```
    --------Testing Error Rate---Training Accuracy---Test Error Rate---
    Test Accuracy

    Original            0.036           0.964           0.079
    0.921


    3 Layers            0.048           0.952           0.097
    0.903
    (100->30->1)
    4 Layers            0.043           0.957           0.097
    0.903
    (500->100->30->1)
    5 Layers            0.048           0.952           0.098
    0.902
    (500->100->50->30->1)


    (Training over the same data set n number of times)
    2 Epoch             0.016           0.984           0.057
    0.943
    3 Epoch             0.014           0.986           0.069
    0.931
    4 Epoch             0.001           0.999           0.071
    0.929


    0.010 LR            0.039           0.961           0.082
    0.918
    0.004 RL            0.033           0.967           0.084
    0.916
    0.001 LR            0.078           0.922           0.113
    0.887
```

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?

ANSWER:

The best accuracy I was able to achive with the training data is 0.999 accuracy with 4 epoch (training over the same data set 4 times).

When changing the other hyperparameters, I got both an increase and decrease in training data accuracy but mostly a decrease. Each increase in the number of epoch increased my accuracy and so did slightly alter the learning rate from the default. But, when changing the learning rate too far, I got decreased accuracy. Also, adding more layers/hidden units did not increase my accuracy at all.

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?

ANSWER:

The best accuracy I was able to achive with the testing data is 0.057 accuracy with 2 epoch (training over the same data set 2 times).

We see a reduced accuracy with the testing data with increasing epoch as mentioned in class alongside/despite the increased accuracy of the training data set. But increasing the number of epoch seemed to decrease the testing accuracy implying the possibility of overfitting of the training data was occuring. Each alteration of the learning rate as well as of the number of layers/hidden units decreases the testing accuracy as well.

Part (c) -- 4 pt

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

ANSWER:

The best model hyperparameters one should use is the one that best improved the testing data accuracy (b). This more likely results in your AI program actually identifying whatever it is that you want it to rather than it just memorizing (overfitting) the data set that you have been training it on.