# Lab 1. PyTorch and ANNs

**Deadline**: Monday, Jan 25, 5:00pm.

**Total**: 30 Points

**Late Penalty**: There is a penalty-free grace period of one hour past the deadline. Any work that is submitted between 1 hour and 24 hours past the deadline will receive a 20% grade deduction. No other late work is accepted. Quercus submission time will be used, not your local computer time. You can submit your labs as many times as you want before the deadline, so please submit often and early.

**Grading TA**: Justin Beland, Ali Khodadadi

This lab is based on assignments developed by Jonathan Rose, Harris Chan, Lisa Zhang, and Sinisa Colic.

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://docs.scipy.org/doc/numpy/reference/)](https://docs.scipy.org/doc/numpy/reference/)
- [https://pytorch.org/docs/stable/torch.html (https://pytorch.org/docs/stable/torch.html)](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.**

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [3]:
```
%%shell
jupyter nbconvert --to html /content/drive/MyDrive/APS360_Labs/LAB_1/Lab_1_PyTorch_and_ANNs.ipynb
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/APS360_Labs/LAB_1/Lab_1_PyTorch_and_ANNs.ipynb to html
[NbConvertApp] Writing 582598 bytes to /content/drive/MyDrive/APS360_Labs/LAB_1/Lab_1_PyTorch_and_ANNs.html
```

Out[3]:

## Colab Link

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

Colab Link: https://drive.google.com/file/d/1vuN7y2J_gLzY3PKAEaUyvYnSClA2Y9Iz/view?usp=sharing (https://drive.google.com/file/d/1vuN7y2J_gLzY3PKAEaUyvYnSClA2Y9Iz/view?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/ (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 [ ]:
```python
def sum_of_cubes(n):
    if n>0 and type(n) == int:
      sum = 0
      for i in range(1,n+1):
        sum = sum + i**3
      return print(sum)
    else:
      print("Invalid input")
      return -1

sum_of_cubes(3)
sum_of_cubes(1)
sum_of_cubes(1.01)
sum_of_cubes(-100)
```

```
36
1
Invalid input
Invalid input
```

Out[ ]: -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](https://docs.python.org/3.6/library/stdtypes.html#str.split) (https://docs.python.org/3.6/library/stdtypes.html#str.split)

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

In [30]:
```
def word_lengths(sentence):
    words = sentence.split(' ')
    count = []
    for i in range(len(words)):
      wLength = len(words[i])
      count.append(wLength)
    return print(count)

word_lengths("welcome to APS360!")
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 [39]:
```
def all_same_length(sentence):
    words = sentence.split(' ')
    count = []
    for i in range(len(words)):
      wLength = len(words[i])
      count.append(wLength)
    for i in range(len(count)-1):
      if count[i] == count[i+1]:
        continue
      else:
        return print('False')
    return print('True')

all_same_length("all same length")
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 [40]:
```
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 [41]:
```
matrix = np.array([[1., 2., 3., 0.5],
                   [4., 5., 0., 0.],
                   [-1., -2., 1., 1.]])
vector = np.array([2., 0., 1., -2.])
```

In [42]:
```
matrix.size # Number of elements in the array i.e. the product of the array's dimensions. (here 3x4)
```

Out[42]: 12

In [43]:
```
matrix.shape # Tuple containing array dimensions. (here 3 dimensions, and each dimension has 4 elements)
```

Out[43]: (3, 4)

In [44]:
```
vector.size # Number of elements in the array i.e. the product of the array's dimensions. (here 1x4)
```

Out[44]: 4

In [46]:
```
vector.shape # Tuple containing array dimensions. (here 4 dimensions, and each dimension has 0 elements --> 1 dimesion array)
```

Out[46]: (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

In [144]:
```
output = np.array([0.0, 0.0, 0.0])
for i in range(len(matrix)): # iterate through columns of matrix
   for j in range(len(vector)): # iterate through elements of vector
      output[i] += matrix[i][j] * vector[j]
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 [57]:
```
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 [61]:
```
if output.all() == output2.all():
  print('Outputs match!')
else:
  print('Epic fail')
```

```
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:

In [68]:
```
import time

start_time = time.time() # record the time before running code
# Code from part (b)
output = np.array([.0, .0, .0])
for i in range(len(matrix)):
   for j in range(len(vector)):
     output[i] += matrix[i][j] * vector[j]
end_time = time.time() # record the time after the code is run
diffB = end_time - start_time # compute the difference

start_time = time.time() # record the time before running code
# Code from part (c)
output2 = np.dot(matrix, vector)
end_time = time.time() # record the time after the code is run
diffC = end_time - start_time # compute the difference

resultB = 'Time taken to run part B code: {}'.format(diffB)
print(resultB)
resultC = 'Time taken to run part C code: {}'.format(diffC)
print(resultC)

if diffC < diffB:
  print('Therefore code from part C is faster!')
else:
  print('therefore code from part B is faster!')
```

```
Time taken to run part B code: 0.00026297569274902344
Time taken to run part C code: 7.891654968261719e-05
Therefore code from part C is faster!
```

# 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 [70]:
```
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 (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 [97]:
```
img = plt.imread('https://drive.google.com/uc?export=view&id=1oaLVR2hr1_qzpKQ47i9rVUIklwbDcews')
```

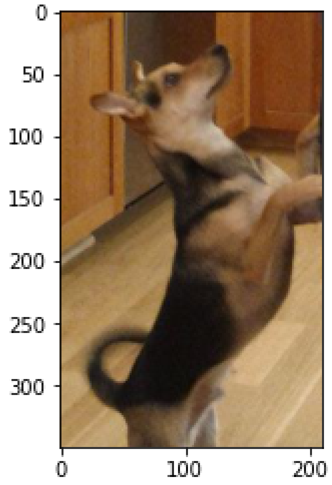
## Part (b) -- 1pt

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 [78]:
```
plt.imshow(img)
```

Out[78]: <matplotlib.image.AxesImage at 0x7f74624ba358>

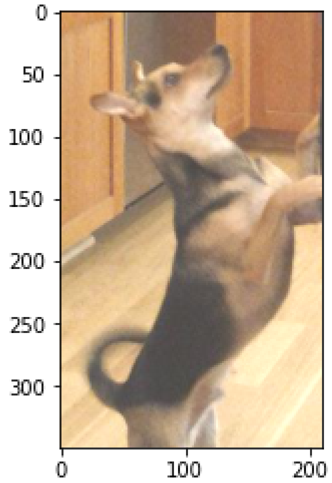



## Part (c) -- 2pt

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 [84]:
```
img_add = img + 0.25
img_add = np.clip(img_add, 0, 1)
plt.imshow(img_add)
```

Out[84]: <matplotlib.image.AxesImage at 0x7f745fe60d68>



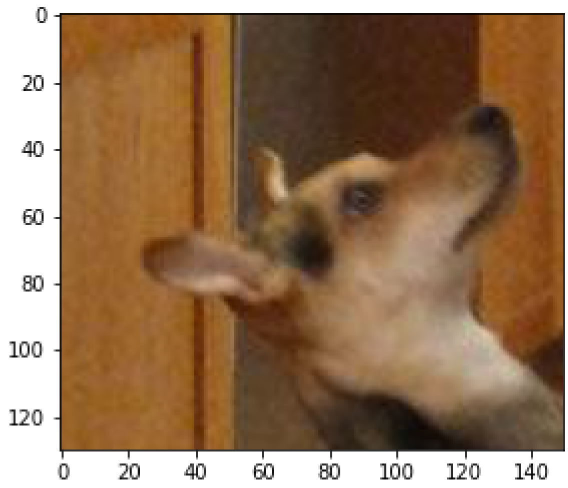

## Part (d) -- 2pt

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 [143]:
```
img_cropped = img[0:130, 0:150, 0:3]
plt.imshow(img_cropped)
```

Out[143]: <matplotlib.image.AxesImage at 0x7f74136a6438>




# 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 [105]:
```
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 [145]:
```
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` .

In [146]:
```
img_torch.shape
```

Out[146]: torch.Size([130, 150, 3])

## Part (c) -- 1pt

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

In [147]:
```
torch.numel(img_torch)
```
Out[147]:
```
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 [148]:
```
# torch.transpose(input, dim0, dim1) → Tensor
# Returns a tensor that is a transposed version of input. The given dimensions dim0 a
nd dim1 are swapped.
trans = img_torch.transpose(0,2)

# we can see from the sizes  of the two tensors that img_torch is NOT updated
sizeTrans = 'Size of transpose of img_torch: {}'.format(trans.shape)
print(sizeTrans)
sizeOriginal ='Size of img_torch: {}'.format(img_torch.shape)
print(sizeOriginal)
```
```
Size of transpose of img_torch: torch.Size([3, 150, 130])
Size of img_torch: torch.Size([130, 150, 3])
```

## 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 [149]:
```
# torch.unsqueeze(input, dim) → Tensor
# Returns a new tensor with a dimension of size one inserted at the specified positio
n.
unsqz = img_torch.unsqueeze(0)

checkSize = 'Size of Unsqueezed Tensor is {} while size of Original Tensor is {}'.for
mat(unsqz.shape, img_torch.shape)
print(checkSize)
if img_torch.shape != unsqz.shape:
  print('Therefore img_torch is not updated')
else:
  print('Therfore img_torch is updated')
```
```
Size of Unsqueezed Tensor is torch.Size([1, 130, 150, 3]) while size of Original Ten
sor is torch.Size([130, 150, 3])
Therefore img_torch is not updated
```

## 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 [159]:
```
max_colors = torch.tensor([0.0, 0.0, 0.0])
for i in range(3):
  max_col = torch.max(img_torch[:,:,i])
  max_colors[i] = max_col
print(max_colors)
```

```
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

In [187]:

```
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, 15)
        self.layer3 = nn.Linear(15, 1) # EXTRA LAYER ADDED HERE
    def forward(self, img):
        flattened = img.view(-1, 28 * 28)
        activation1 = self.layer1(flattened)
        activation1 = F.relu(activation1)
        activation2 = self.layer2(activation1)
        activation3 = self.layer3(activation2)
        return activation3

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=0.009, momentum=0.9) #LEARNING RATE MOD
IFIED HERE
for training_iteration in range(20): # EXTRA TRAINING ITERATIONS ADDED HERE
    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
ameter)
        optimizer.step()                   # step 4 (make the updates for each parame
ter)
        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: 0.0
Training Accuracy: 1.0
Test Error Rate: 0.047
Test Accuracy: 0.953
```

### 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?

By adding 10 training iterations and decreasing the learning rate to 0.003 I was able to get **100%** Training Accuracy and zero Error Rate. The Testing accuracy however, only reached 93.3%, probaly due to training overfitting. In addition, I was able to reach the same training accuracy of 100% by adding a third layer and 20 training iterations (learing rate same as given from template: 0.005). In this second case Test accuracy is also increased to 94%, which makes me conclude that this second way is better than the first one.

### 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?

The best Testing Accuracy I was able to get was **95.3%** and I achieved that by adding a third layer and 20 training iterations as in part (a) but I aslo increased the learning rate to 0.009. In this way I also mantained 100% Training accuracy as shown from the code output.\ By experiments I have made with the code I have also noticed that "more" is not necessarly "better"; indeed, adding a fouth layer or increasing the number of training iterations even more does not produce a higher testing performance. Finally, using the ReLU activation function did not give a higher testing performance as I was expecting; this perhaps might be because our neural network is not deep enough to see a difference from a sigmoid or linear activation function.

### Part (c) -- 4 pt

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

Hyperparamenters from part (b) are recommended since the performance in a real world dataset scenario is better represented by the testing set rather than the training set.\ Indeed, very high performances can always be reached in the training process by overfitting on the training data. But training accurancy does not guarantee at all that, given a random dataset, we will get the same high preformance; in fact, most of the times it occurs the opposite.