Lab 1. PyTorch and ANNs

Deadline: Thursday, May 21, 11:59pm.

Total: 30 Points

Late Penalty: There is a penalty-free grace period of one hour past the deadline. Any work that is s
the deadline will receive a 20% grade deduction. No other late work is accepted. Quercus submiss
time. You can submit your labs as many times as you want before the deadline, so please submit (

Grading TA: Justin Beland

This lab is partially based on an assignment developed by Prof. Jonathan Rose and Harris Chan.

This lab is a warm up to get you used to the PyTorch programming environment used in the course
knowledge of Python and relevant Python libraries. The lab must be done individually. Please recal
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
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, pleas
Jupyter Notebook file to Google Colab for submission.

With Colab, you can export a PDF file using the menu option `File -> Print` and save as PDF file.

Colab Link

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

Colab Link: https://colab.research.google.com/drive/1Eb914nVXR-23vGy-3NHNWpboWkUEdc3G?

Part 1. Python Basics [3 pt]

The purpose of this section is to get you used to the basics of Python, including working with func

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-ı

Part (a) -- 1pt

Write a function `sum_of_cubes` that computes the sum of cubes up to `n`. If the input to `sum_of_cu` 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
    """
    result = 0
    if (n < 0) or not(isinstance(n,int)):
      print("Invalid input")
      result = -1
    else:
      for i in range(n+1):
        result += i**3

    return result

# print (sum_of_cubes(3))
```

Part (b) -- 1pt

Write a function `word_lengths` that takes a sentence (string), computes the length of each word ir 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 check out https://docs.python.org/3.6/library/stdtypes.html#str.split

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

```
Help on method_descriptor:

split(...)
    S.split(sep=None, maxsplit=-1) -> list of strings

    Return a list of the words in S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator and empty strings are
```

```
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]
    """
    strLen = [];
#    if sentence == "" or sentence ==" ":
#      strLen.append(0)
    for i in sentence.split():
      strLen.append(len(i))

    return strLen

print(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 i
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
    """
    result = False
#    if (word_lengths(sentence) == [0]):
#      result = False
    if (len(set(word_lengths(sentence))) == 1):
      result = True


    return result

print(all_same_length("hello world"))
```

```
print(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 sh
`numpy`.

```
import numpy as np
```

Part (a) -- 1pt

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

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

```
matrix.shape
```

```
(3, 4)
```

Shape of an NumpyArray will give a tuple that reflects the number of dimension and number of ele

- (#rows, #columes) ie. matrix will have 3 dimensions and 4 elements in each dimension, 3 ro

```
vector.size
```

```
4
```

Size of an NumpyArray will give an integer that reflects the total number of elements in the array.

- ie. this vector has size of 4

```
vector.shape
```

```
(4,)
```

Since a vector only has 1 row, (4,) means this vector has 4 columns

- (#columes,)

Part (b) -- 1pt

Perform matrix multiplication `output = matrix x vector` by using for loops to iterate through the
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
mrow, mcol = matrix.shape
output = np.zeros(matrix.shape[0], dtype=float)  # output = np.array(output)
for i in range(mrow):
  for j in range(mcol):
    output[i] +=  matrix[i][j] * vector[j]

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 ea `numpy.dot` is much faster (it is written in C and highly optimized). In general, we will avoid for loop

```
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("outputs match")
else:
  print("outputs not match")
```

```
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):
    99*99

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

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

```
0.0004963874816894531
```

```
start1 = time.time()
mrow, mcol = matrix.shape
output = np.zeros(matrix.shape[0], dtype=float)  # output = np.array(output)
for i in range(mrow):
  for j in range(mcol):
    output[i] +=  matrix[i][j] * vector[j]
end1 = time.time()
diff1 = end1 - start1

start2 = time.time()
output2 = np.dot(matrix, vector)
output2
end2 = time.time()
diff2 = end2 - start2

if diff1 > diff2: print("np.dot is faster")
else: print("for loop is faster")
```

```
np.dot 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, wl
width of the image, and C is the number of colour channels. Typically we will use an image with ch
“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

```
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_qzpKQ47i9
the `plt.imread` function.

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

```
img = plt.imread("https://drive.google.com/uc?export=view&id=1oaLVR2hr1_qzpKQ47i9rVUIklwbD
```

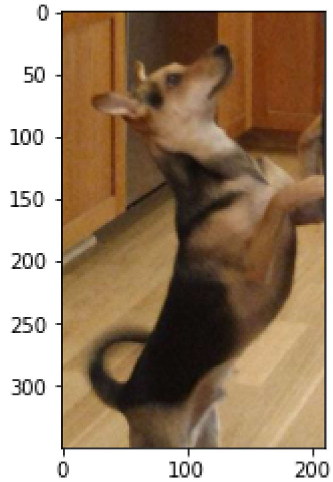
Part (b) -- 1pt

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 l
the Y (row) direction, and the second dimension indicates the X (column) dimension.

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

```
<matplotlib.image.AxesImage at 0x7f208788c2b0>
```

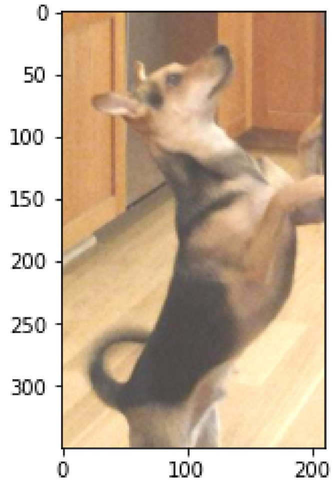


Part (c) -- 2pt

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

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

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
<matplotlib.image.AxesImage at 0x7f20886a6198>
```



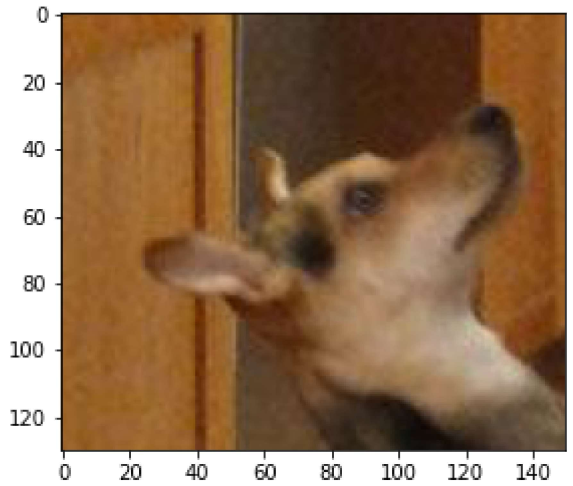
Part (d) -- 2pt

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

Display the image.

```
img_cropped = img[:130, :150, :3] # crop first 130x150 of the matrix
plt.imshow(img_cropped)
```

```
<matplotlib.image.AxesImage at 0x7f2087c8e588>
```



Part 4. Basics of PyTorch [6 pt]

PyTorch is a Python-based neural networks package. Along with tensorflow, PyTorch is currently o
libraries.

PyTorch, at its core, is similar to Numpy in a sense that they both try to make it easier to write code
performance over vanilla Python by leveraging highly optimized C back-end. However, compare to
support and provides many high-level features for machine learning. Technically, Numpy can be us
does. However, Numpy would be a lot slower than PyTorch, especially with CUDA GPU, and it woul
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 tenso
`img_torch`.

```
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`?

```
if img_torch.is_floating_point():
  result = img_torch.shape[0] * img_torch.shape[1] * img_torch.shape[2]
  print(result)
else:
  print("torch is not float type")
```

```
58500
```

Part (d) -- 1 pt

What does the code `img_torch.transpose(0,2)` do? What does the expression return? Is the origi

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

```
tensor([[[0.5882, 0.5412, 0.6157,  ..., 0.6039, 0.5882, 0.5804],
         [0.5765, 0.5647, 0.6196,  ..., 0.6078, 0.6078, 0.6039],
         [0.5569, 0.5961, 0.6196,  ..., 0.6118, 0.6196, 0.6235],
         ...,
         [0.5804, 0.5882, 0.5922,  ..., 0.3804, 0.3882, 0.4196],
         [0.6039, 0.6078, 0.6157,  ..., 0.3765, 0.3804, 0.4039],
         [0.6157, 0.6196, 0.6275,  ..., 0.3765, 0.3804, 0.3961]],

        [[0.3725, 0.3216, 0.3765,  ..., 0.3882, 0.3725, 0.3647],
         [0.3608, 0.3451, 0.3843,  ..., 0.3922, 0.3922, 0.3882],
         [0.3412, 0.3765, 0.3843,  ..., 0.3961, 0.4039, 0.4078],
         ...,
         [0.3412, 0.3490, 0.3529,  ..., 0.3098, 0.3176, 0.3373],
         [0.3647, 0.3686, 0.3765,  ..., 0.3059, 0.3098, 0.3216],
         [0.3765, 0.3804, 0.3882,  ..., 0.3098, 0.3098, 0.3137]],

        [[0.1490, 0.0902, 0.1529,  ..., 0.1686, 0.1529, 0.1451],
         [0.1373, 0.1137, 0.1490,  ..., 0.1686, 0.1725, 0.1686],
         [0.1176, 0.1451, 0.1412,  ..., 0.1725, 0.1804, 0.1882],
         ...,
         [0.1294, 0.1373, 0.1373,  ..., 0.2157, 0.2314, 0.2549],
         [0.1529, 0.1569, 0.1608,  ..., 0.2118, 0.2157, 0.2392],
         [0.1647, 0.1686, 0.1725,  ..., 0.2078, 0.2157, 0.2314]]])
```

The tranpose of `img_torch`, dimension 0 and 2 are swapped. The original variable is not updated, s
transpose version.

Part (e) -- 1 pt

What does the code `img_torch.unsqueeze(0)` do? What does the expression return? Is the origina

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

```
tensor([[[[0.5882, 0.3725, 0.1490],
          [0.5765, 0.3608, 0.1373],
          [0.5569, 0.3412, 0.1176],
          ...,
          [0.5804, 0.3412, 0.1294],
          [0.6039, 0.3647, 0.1529],
          [0.6157, 0.3765, 0.1647]],

         [[0.5412, 0.3216, 0.0902],
          [0.5647, 0.3451, 0.1137],
          [0.5961, 0.3765, 0.1451],
          ...,
          [0.5882, 0.3490, 0.1373],
          [0.6078, 0.3686, 0.1569],
          [0.6196, 0.3804, 0.1686]],

         [[0.6157, 0.3765, 0.1529],
          [0.6196, 0.3843, 0.1490],
          [0.6196, 0.3843, 0.1412],
          ...,
          [0.5922, 0.3529, 0.1373],
          [0.6157, 0.3765, 0.1608],
          [0.6275, 0.3882, 0.1725]],

         ...,

         [[0.6039, 0.3882, 0.1686],
```

It will return a new tensor with a dimension of size one inserted at index 0 (from input = 0). Origina
tendor shares the same underlaying data with the original one.

Part (f) -- 1 pt

Find the maximum value of `img_torch` along each colour channel? Your output should be a one-di
values.

Hint: lookup the function `torch.max`.

```
img_torch.max(0).values.max(0).values
```

```
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 l
Modify the code by changing any of the following and observe how the accuracy and error are affe

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

```python
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)
        #self.layer3 = nn.Linear(15,1)
    def forward(self, img):
        flattened = img.view(-1, 28 * 28)
        activation1 = self.layer1(flattened)
        activation1 = F.relu(activation1)
        #activation1 = F.tanh(activation1)
        activation2 = self.layer2(activation1)
        #activation2 = F.relu(activation2)
        #activation3 = self.layer3(activation2)
        #return activation3
        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=0.008, momentum=0.9)

#inter_num = 10
#for i in range(inter_num):
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))
```

```
Training Error Rate: 0.038
Training Accuracy: 0.962
Test Error Rate: 0.088
Test Accuracy: 0.912
```

Part (a) -- 3 pt

Comment on which of the above changes resulted in the best accuracy on training data? What acc

Change in number of iterations will result in the best training arrucary: highest training accuracy =

- Original:
  - Error Rate: 0.036
  - Training Accuracy: 0.964
- Increase #iterations: 1 to 10
  - Training Error Rate: 0.001
  - Training Accuracy: 0.999
- Increase #hidden units: 30 to 100
  - Training Error Rate: 0.03
  - Training Accuracy: 0.97
- Decrease #hidden units: 30 to 10
  - Training Error Rate: 0.047
  - Training Accuracy: 0.953
- Increase #layers: 2 to 3
  - Training Error Rate: 0.041
  - Training Accuracy: 0.959
- Change activation function: use tanh

- Training Error Rate: 0.04
    - Training Accuracy: 0.96
- Increase learning rate: 0.005 to 0.008
    - Training Error Rate: 0.038
    - Training Accuracy: 0.962
- Decrease learning rate: 0.005 to 0.001
    - Training Error Rate: 0.078
    - Training Accuracy: 0.922

Part (b) -- 3 pt

Comment on which of the above changes resulted in the best accuracy on testing data? What acc

Change in number of iterations will result in the best testing arrucary: highest testing accuracy = 0

- Original
    - Test Error Rate: 0.079
    - Test Accuracy: 0.921
- Increase #iterations: 1 to 10
    - Test Error Rate: 0.059
    - Test Accuracy: 0.9410000000000001
- Increase #hidden units: 30 to 100
    - Test Error Rate: 0.077
    - Test Accuracy: 0.923
- Decrease #hidden units: 30 to 10
    - Test Error Rate: 0.104
    - Test Accuracy: 0.896
- Increase #layers: 2 to 3
    - Test Error Rate: 0.1
    - Test Accuracy: 0.9
- Change activation function: use tanh
    - Test Error Rate: 0.094
    - Test Accuracy: 0.906
- Increase learning rate: 0.005 to 0.008
    - Test Error Rate: 0.088
    - Test Accuracy: 0.912

- Decrease learning rate: 0.005 to 0.001
    - Test Error Rate: 0.113
    - Test Accuracy: 0.887

Part (c) -- 4 pt

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

- Model hyperparameters in (b) should be used.
- Use hyperparameters from traning accuracy may result in high traning accuracy and low test
- In that case, NN will only learn the rules from training set, but not all the rules can be applied