Lab 1. PyTorch and ANNs

Deadline: Thursday, January 23, 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: Kevin Course

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 cours
knowledge of Python and relevant Python libraries. The lab must be done individually. Please reca
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/11qxKJ6RsgOOtbTbyA0Wg-h6-_pmSwHS6

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-

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
    """
    if (type(n) != int or n <= 0):
        print("Invalid input")
        return -1


    # Formula for sum of first n cubes:
    return ((n * (n+1)) / 2) ** 2
```

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
    removed from the result.
```

```
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]
    """
    return [len(word) for word in sentence.split(" ")]
```

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
    """

    lengths = word_lengths(sentence)
    return lengths.count(lengths[0]) == len(lengths)
```

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>.`

Answer to Part 2(a):

`<NumpyArray>.size` represents the number of elements in the numpy array.

`<NumpyArray>.shape` represents the number of rows and columns in the numpy array.

```
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
```

```
12
```

```
matrix.shape
```

```
(3, 4)
```

```
vector.size
```

```
4
```

```
vector.shape
```

```
(4,)
```

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

```
def loopMultiply():
  ret = []
  for i in range(matrix.shape[0]):
    sum = 0
    for j in range(matrix.shape[1]):
      sum += matrix[i][j] * vector[j];
    ret.append(sum)

  return np.array(ret)

output = loopMultiply()
```

```
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

```
def npDot():
  return np.dot(matrix, vector)

output2 = npDot()


output2
```

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

Part (d) -- 1pt

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

```
np.array_equal(output, output2)
```

```
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:

```
import time

def timeTaken(func):
  # record the time before running code
  start_time = time.time()

  # place code to run here
  for i in range(10000):
      func()

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

  # compute the difference
  return end_time - start_time


t1 = timeTaken(loopMultiply)
```

```
t2 = timeTaken(npDot)

print("Time taken for for loop multiplication method: " + str(t1) + "s")
print("Time taken for np multiplication: " + str(t2) + "s")
print("np multiplication is " + str(t1 - t2) + "s faster")
```

```
Time taken for for loop multiplication method: 0.12101387977600098s
Time taken for np multiplication: 0.012632369995117188s
np multiplication is 0.10838150978088379s faster
```

Part 3. Images [6 pt]

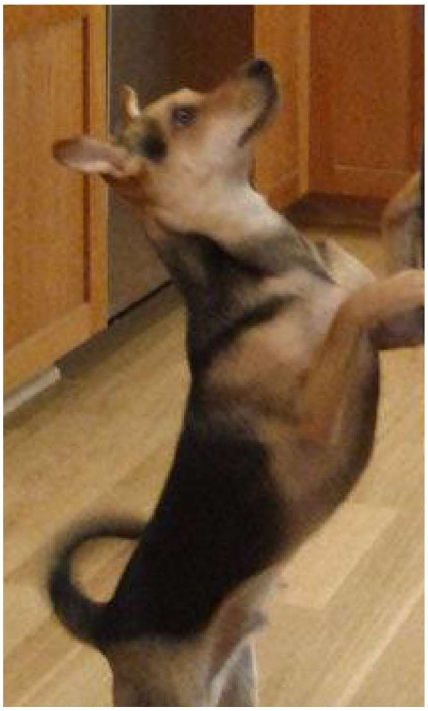
A picture or image can be represented as a NumPy array of “pixels”, with dimensions H × W × C, w
width of the image, and C is the number of colour channels. Typically we will use an image with ch
“level” of each pixel, which is referred to with the short form RGB.

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_qzpKQ47i
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
```

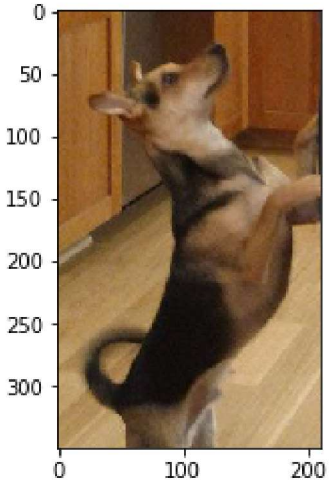
Part (b) -- 1pt

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 0x7f8900359f60>
```

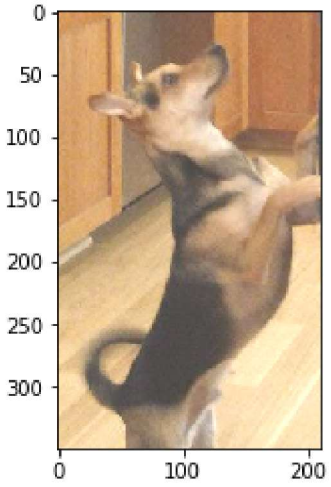


Part (c) -- 2pt

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 rang
value that is outside of the desired range to the closest endpoint. Display the image using `plt.ims`

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

```
<matplotlib.image.AxesImage at 0x7f88fdff4fd0>
```

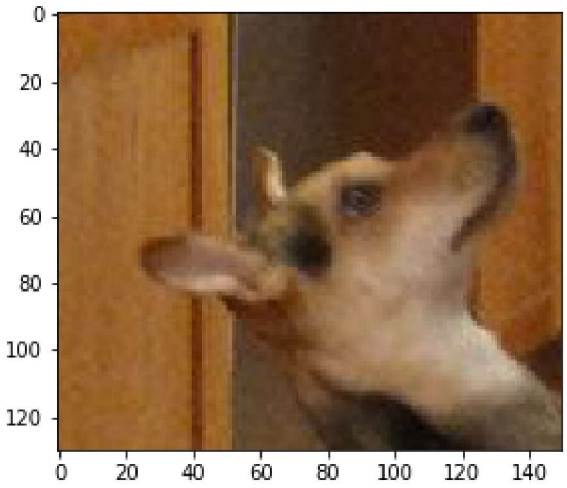


Part (d) -- 2pt

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**)

```
img_cropped = np.delete(np.asarray(img), np.s_[150:], axis=1)   # crop width to 150
img_cropped = np.delete(np.asarray(img_cropped), np.s_[130:], axis=0)   # crop height to 1
img_cropped = np.delete(np.asarray(img_cropped), 3, axis=2)   # remove alpha channel
plt.imshow(img_cropped)
```

```
<matplotlib.image.AxesImage at 0x7f88fdef1e10>
```



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 cod 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 tensc `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`?

```
sz = 1
for i in img_torch.shape:
  sz *= i

sz
```

```
58500
```

Part (d) -- 1 pt

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

Answer for Part 4(d)

The code `img_torch.transpose(0,2)` transposes the first the third column of `img_torch`. It return `img_torch` is unaffected.

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

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

Answer for Part 4(e)

The code `img_torch.unsqueeze(0)` adds a new dimension to the array `img_torch` at index `0`. It re `img_torch` is unaffected.

```
img_unsq = img_torch.unsqueeze(0)
print(img_torch.shape)
print(img_unsq.shape)
```

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

Hint: lookup the function `torch.max`.

```
vals = torch.max(img_torch.transpose(0, 2), 2)[0]
vals = torch.max(vals, 1)[0]
vals
```

```
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

```
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

num_training_iterations = 20
learning_rate = 0.01

# 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=learning_rate, momentum=0.9)

for i in range(num_training_iterations):
  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: 0.0
Training Accuracy: 1.0
Test Error Rate: 0.055
Test Accuracy: 0.945
```

Part (a) -- 3 pt

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

Answer to Part 5(a)

The best training data accuracy I could reach is 100% with a sufficiently large number of training it
iterations achieves better training results. However, this can lead to overfitting to training data and

Tweaked the learning rate only affects how many iterations are required to reach a high accuracy r

Part (b) -- 3 pt

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

Answer to Part 5(a)

The best testing data accuracy I could reach is 94.5% with number of training iterations = 20 and l
number of training iterations causes a fall in testing accuracy due to overfitting to the training data
high also affects the testing data accuracy negatively.

Part (c) -- 4 pt

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

Answer to Part 5(a)

We should use the model hyperparameters from part (b) as the parameters from part (a) overfit to
outside the training data.