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/15C3LpwYxoaTnAlBrJVm-B-4BwufrydMO?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`.

In [166…

```
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
    """
    sum = 0
    if(n <= 0):
        print("Invalid input")
        return -1
    else:
        for i in range(0,n):
          i += 1
          sum += i**3

    return sum
```

In [167…
```
sum_of_cubes(1)
```
Out[167]:
```
1
```

In [168…
```
sum_of_cubes(0)
```
```
Invalid input
```
Out[168]:
```
-1
```

In [169…
```
sum_of_cubes(3)
```
Out[169]:
```
36
```

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 [170…
```
help(str.split)
```
```
Help on built-in function split:

split(sep=None, maxsplit=-1) method of builtins.str instance
    Return a list of the substrings in the string, using sep as the separato
r 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.
```

In [171…
```
str = 'hello world'
str.split()
```

Out[171]:
```
['hello', 'world']
```

In [172…
```
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]
    """
    lengths = []

    words = sentence.split()
    count = 0
    for i in range(len(words)):
      count = len(words[i])
      lengths.append(count)

    return lengths
```

In [173…
```
word_lengths("welcome to APS360!")
```

Out[173]:
```
[7, 2, 7]
```

In [174…
```
word_lengths("machine learning is so cool")
```

Out[174]:
```
[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 [175...
```python
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
    """
    length = word_lengths(sentence)
    length_1 = length[0]
    isSame = False
    for numbers in length:
      if(numbers != length_1):
        return False
      else:
        isSame = True
        continue

    return isSame
```

In [176...
```python
all_same_length("all same length")
```

Out[176]:
```
False
```

In [177...
```python
all_same_length("Hello World")
```

Out[177]:
```
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 [178...
```python
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 [179...
```python
matrix = np.array([[1., 2., 3., 0.5],
                   [4., 5., 0., 0.],
                   [-1., -2., 1., 1.]])
vector = np.array([2., 0., 1., -2.])
```

In [180…
```
matrix.size
```
Out[180]:
```
12
```

In [181…
```
matrix.shape
```
Out[181]:
```
(3, 4)
```

In [182…
```
vector.size
```
Out[182]:
```
4
```

In [183…
```
vector.shape
```
Out[183]:
```
(4,)
```

Answer: `<NumpyArray>.size` gives us how many elements are there in a `NumpyArray` `<NumpyArray>.shape` returns a tuple, giving us the dimension of the `NumpyArray`. The first element represents how many rows there are, the second element represents how many columns there are.

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 [184…
```
result = []

for i in range(len(matrix)):
  sum = 0
  for j in range(len(vector)):
    sum += vector[j] * matrix[i][j]
  result.append(sum)

output = np.array(result)
```

In [185…
```
output
```
Out[185]:
```
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.

In [186…

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

In [187…

```
output2
```

Out[187]:

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

Part (d) -- 1pt

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

In [188…

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

```
The two 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 [189…

```
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
```

Out[189]:

```
0.0009312629699707031
```

In [190…

```
output1_start_time = time.time()

result = []

for i in range(len(matrix)):
  sum = 0
  for j in range(len(vector)):
    sum += vector[j] * matrix[i][j]
  result.append(sum)

output = np.array(result)

output1_end_time = time.time()
diff1 = output1_end_time - output1_start_time

output2_start_time = time.time()

output2 = np.dot(matrix, vector)

output2_end_time = time.time()
diff2 = output2_end_time - output2_start_time

if(diff2 < diff1):
  print("np.dot is faster.")
else:
  print("Code in part(b) 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, 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 [191…

```
import matplotlib.pyplot as plt
import urllib
import PIL
import numpy as np
```

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 [192…

```
img = np.array(PIL.Image.open(urllib.request.urlopen("https://drive.google.c
```

Part (b) -- 1pt

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

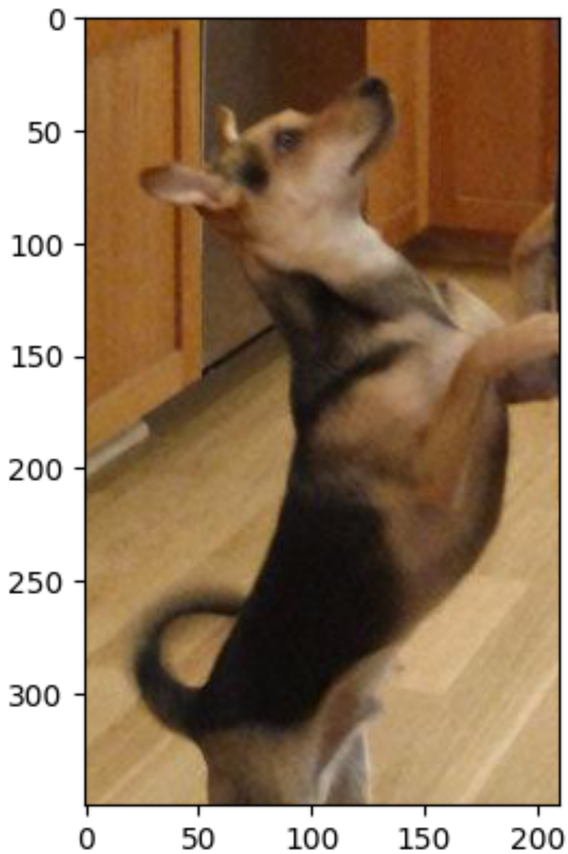
This function will also show the coordinate system used to identify pixels. The origin is at the top left corner, and the first dimension indicates the Y (row) direction, and the second dimension indicates the X (column) dimension.

In [193…

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

Out[193]:

```
<matplotlib.image.AxesImage at 0x7efe5187d660>
```



Part (c) -- 2pt

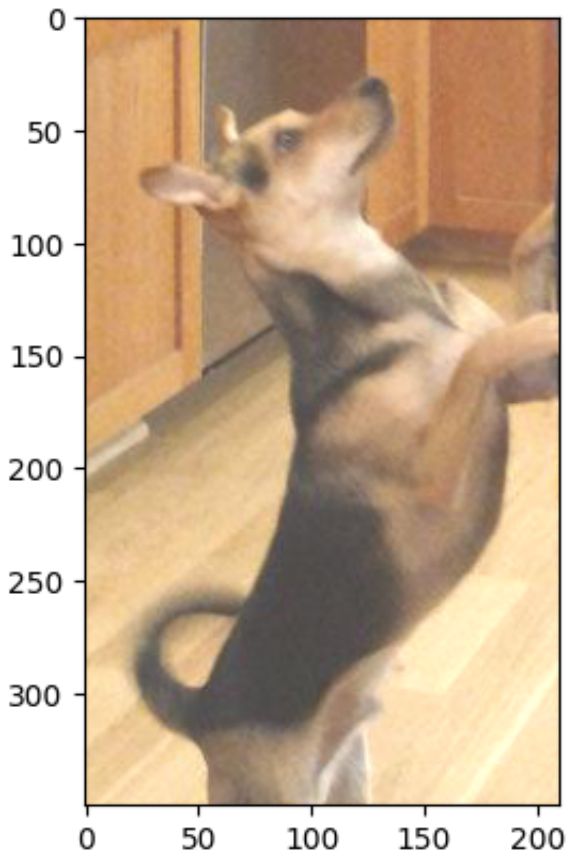
Modify the image by adding a constant value of 0.25 to each pixel in the `img` and store the result in the variable `img_add`. Note that, since the range for the pixels needs to be between [0, 1], you will also need to clip img_add to be in the range [0, 1] using `numpy.clip`. Clipping sets any value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

In [194…

```
img_add = np.clip(img/255 + 0.25,0,1)
plt.imshow(img_add)
```

Out[194]:

```
<matplotlib.image.AxesImage at 0x7efe519f12d0>
```



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 [195…
```
img_cropped = img[:130,:150,:3]
plt.imshow(img_cropped)
```

Out[195]:
```
<matplotlib.image.AxesImage at 0x7efe60aed150>
```

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 [196…

```
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 [197…
```
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 [198…
```
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 [199…
```
print(img_torch, '\n')
print("No floating-point numbers are stored in the tensor img_torch. All ele
```

```
tensor([[[150,  95,  38],
         [147,  92,  35],
         [142,  87,  30],
         ...,
         [148,  87,  33],
         [154,  93,  39],
         [157,  96,  42]],

        [[138,  82,  23],
         [144,  88,  29],
         [152,  96,  37],
         ...,
         [150,  89,  35],
         [155,  94,  40],
         [158,  97,  43]],

        [[157,  96,  39],
         [158,  98,  38],
         [158,  98,  36],
         ...,
         [151,  90,  35],
         [157,  96,  41],
         [160,  99,  44]],

        ...,

        [[154,  99,  43],
         [155, 100,  43],
         [156, 101,  44],
         ...,
         [ 97,  79,  55],
         [ 96,  78,  54],
         [ 96,  79,  53]],

        [[150,  95,  39],
         [155, 100,  44],
         [158, 103,  46],
         ...,
         [ 99,  81,  59],
         [ 97,  79,  55],
         [ 97,  79,  55]],

        [[148,  93,  37],
         [154,  99,  43],
         [159, 104,  48],
         ...,
         [107,  86,  65],
         [103,  82,  61],
         [101,  80,  59]]], dtype=torch.uint8)
```

```
No floating-point numbers are stored in the tensor img_torch. All elements a
re integers.
```

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 [200…

```
print(img_torch.transpose(0,2))
a = img_torch.transpose(0,2)
print('\n', 'Transposed img_torch dimension: ',a.shape)
print('\n','Current img_torch dimension: ',img_torch.shape)
print("\n", "img_torch.transpose(0,2) returns a tensor that have a transpose
      '\n', "The original variable img_torch is not updated because the func
```

```
tensor([[[150, 138, 157,  ..., 154, 150, 148],
         [147, 144, 158,  ..., 155, 155, 154],
         [142, 152, 158,  ..., 156, 158, 159],
         ...,
         [148, 150, 151,  ...,  97,  99, 107],
         [154, 155, 157,  ...,  96,  97, 103],
         [157, 158, 160,  ...,  96,  97, 101]],

        [[ 95,  82,  96,  ...,  99,  95,  93],
         [ 92,  88,  98,  ..., 100, 100,  99],
         [ 87,  96,  98,  ..., 101, 103, 104],
         ...,
         [ 87,  89,  90,  ...,  79,  81,  86],
         [ 93,  94,  96,  ...,  78,  79,  82],
         [ 96,  97,  99,  ...,  79,  79,  80]],

        [[ 38,  23,  39,  ...,  43,  39,  37],
         [ 35,  29,  38,  ...,  43,  44,  43],
         [ 30,  37,  36,  ...,  44,  46,  48],
         ...,
         [ 33,  35,  35,  ...,  55,  59,  65],
         [ 39,  40,  41,  ...,  54,  55,  61],
         [ 42,  43,  44,  ...,  53,  55,  59]]], dtype=torch.uint8)

 Transposed img_torch dimension:  torch.Size([3, 150, 130])

 Current img_torch dimension:  torch.Size([130, 150, 3])

 img_torch.transpose(0,2) returns a tensor that have a transposed (between i
ts height, width, and color channel dimensions) version of its input.
 The original variable img_torch is not updated because the function returns
a new transposed tensor but does not update the input tensor.
```

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 [201…

```
print(img_torch.unsqueeze(0))
b = img_torch.unsqueeze(0)
print('\n', 'Unsqueezed img_torch dimension: ', b.shape)
print('\n','Current img_torch dimension: ',img_torch.shape)
print('\n', "img_torch.unsqueeze(0) returns a new tensor with a dimension of
print("The original variable img_torch is not updated because the function r
```

```
tensor([[[[150,  95,  38],
          [147,  92,  35],
          [142,  87,  30],
          ...,
          [148,  87,  33],
          [154,  93,  39],
          [157,  96,  42]],

         [[138,  82,  23],
          [144,  88,  29],
          [152,  96,  37],
          ...,
          [150,  89,  35],
          [155,  94,  40],
          [158,  97,  43]],

         [[157,  96,  39],
          [158,  98,  38],
          [158,  98,  36],
          ...,
          [151,  90,  35],
          [157,  96,  41],
          [160,  99,  44]],

         ...,

         [[154,  99,  43],
          [155, 100,  43],
          [156, 101,  44],
          ...,
          [ 97,  79,  55],
          [ 96,  78,  54],
          [ 96,  79,  53]],

         [[150,  95,  39],
          [155, 100,  44],
          [158, 103,  46],
          ...,
          [ 99,  81,  59],
          [ 97,  79,  55],
          [ 97,  79,  55]],

         [[148,  93,  37],
          [154,  99,  43],
          [159, 104,  48],
          ...,
```

```
         [107,  86,  65],
         [103,  82,  61],
         [101,  80,  59]]]], dtype=torch.uint8)

 Unsqueezed img_torch dimension:  torch.Size([1, 130, 150, 3])

 Current img_torch dimension:  torch.Size([130, 150, 3])

 img_torch.unsqueeze(0) returns a new tensor with a dimension of size one in
serted at its height dimension.
The original variable img_torch is not updated because the function returns
a new tensor rather than modifying the input.
```

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 [202…

```
values = []
for i in range(len(img_torch[0][0])):
  values.append(torch.max(img_torch[:,:,i]))

max_values = torch.tensor(values)

print(max_values)
```

```
tensor([228, 201, 172], dtype=torch.uint8)
```

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 [203…

```
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
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 eac
    optimizer.step()                   # step 4 (make the updates for each p
    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: 0.036
Training Accuracy: 0.964
Test Error Rate: 0.079
Test Accuracy: 0.921
```

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?

In [204…

```
from tabulate import tabulate
```

In [205…

```
col_names1 = ['Learning Rates', 'Training Accuracy', 'Test Accuracy']
data1 = [[0.005, 0.964, 0.921],[0.01, 0.961, 0.918], [0.003, 0.966, 0.91]]
print(tabulate(data1, headers = col_names1))
col_names2 = ['Hidden Units', 'Training Accuracy', 'Test Accuracy']
data2 = [[30, 0.964, 0.921], [10, 0.953, 0.896], [50, 0.967, 0.926]]
print('\n', tabulate(data2, headers = col_names2))
col_names3 = ['Activation Function', 'Training Accuracy', 'Test Accuracy']
data3 = [['Relu', 0.964, 0.921],['Sigmoid', 0.927, 0.883], ['Tanh', 0.96, 0.
print('\n', tabulate(data3, headers = col_names3))
print('\n')
print('The above data was obtained from changing parameters from three categ
print('For each category, the changes are only made to the parameters relate
print('\n')
print('When learning rate is changed to 0.003, the model results in the best
print('When hidden unit is changed from 30 to 50, the model results in the b
print('When activation function remain the same as the ReLu function, the mc
```

```
  Learning Rates    Training Accuracy    Test Accuracy
----------------  -------------------  ---------------
           0.005                0.964            0.921
           0.01                 0.961            0.918
           0.003                0.966            0.91

   Hidden Units    Training Accuracy    Test Accuracy
--------------  -------------------  ---------------
             30                0.964            0.921
             10                0.953            0.896
             50                0.967            0.926

 Activation Function      Training Accuracy    Test Accuracy
---------------------  -------------------  ---------------
Relu                                 0.964            0.921
Sigmoid                              0.927            0.883
Tanh                                 0.96             0.906


The above data was obtained from changing parameters from three categories:
learning rates, hidden units, and activation function.
For each category, the changes are only made to the parameters related to th
e category, keeping the rest of the code the same as the original code.


When learning rate is changed to 0.003, the model results in the best accura
cy for training data: 0.966.
When hidden unit is changed from 30 to 50, the model results in the best acc
uract for training data: 0.967.
When activation function remain the same as the ReLu function, the model res
ults in the best accuracy for training data: 0.964.
```

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:

When learning rate remains unchanged as 0.005, the model results in the best accuracy for test data: 0.921.

When hidden unit is changed from 30 to 50, the model results in the best accuract for test data: 0.926.

When activation function remains the same as the ReLu function, the model results in the best accuracy for test data: 0.921.

Part (c) -- 4 pt

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

Answer: We should use hyperparameters from (b) to allow for better accuracy on the test data. Since we would expect our model to perform in real life scneario, making prediction on cases the model has never seen before, a better accuracy on the test data will allow the model to perform the best in real life situations. Also, a high accuracy on training data only means that the model has seen the training data multiple times and captures most correlations in the training data. The model would fit the training data better and better when optimizing the parameters based on training data accuracy, but when it is applied to test data or real life cases, the model may overfit and result in loss of accuracy on test data.