In [8]:
```
# %%shell
# jupyter nbconvert --to html "/Users/harrynguyen/Documents/GitHub/APS360/La
%%shell
jupyter nbconvert --to html "/content/Lab1_PyTorch_and_ANNs.ipynb"
```

```
[NbConvertApp] Converting notebook /content/Lab1_PyTorch_and_ANNs.ipynb to h
tml
[NbConvertApp] WARNING | Alternative text is missing on 3 image(s).
[NbConvertApp] Writing 986460 bytes to /content/Lab1_PyTorch_and_ANNs.html
```

Out[8]:

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:

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 [ ]:
```
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 not isinstance(n, int) or n <= 0:
        print("Invalid input")
        return -1
    return sum(n**3 for i in range (1, n+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 [ ]:
```
help(str.split)
```

```
Help on method_descriptor:

split(self, /, sep=None, maxsplit=-1)
    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 [ ]:
```
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
```

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 [ ]:
```
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
    """
    words = sentence.split()
```

```
    if not words:
        return True  # Handle the edge case of an empty sentence
    first_length = len(words[0])
    return all(len(word) == first_length for word in words)
```

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

In [ ]:
```
matrix.size

#matrix.size returns the the total number of elements in the array
#For the example above, the output is (4 by 3) 12 (elements).
```

Out[ ]:
```
12
```

In [ ]:
```
matrix.shape

#matrix.shape returns a tuple representing the dimensions of the array.
#Generally in 2D, the first element represents the number of rows and the
#second element represents the number of columns.
#For the example above, the output is (3, 4).
```

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

In [ ]:
```
vector.size

#Similarly, vector.size returns the total number of elements in the vector (
#For the example above, the output is 4.
```

Out[ ]:
```
4
```

In [ ]:
```
print(vector.shape)

#vector.shape returns a tuple representing the dimensions of the vector. In
#it will return a tuple with one value, indicating the number of elements.
#For the example above, the output is (4, ).
```

```
(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 [ ]:
```
output = [0] * len(matrix)

#Perform matrix multiplications
for i in range(len(matrix)):
    sum = 0
    for j in range(len(vector)):
        sum += matrix[i][j] * vector[j]
    output[i] = sum

output = np.array(output)

print("Output:", output)
```

```
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 [ ]:
```
output2 = np.dot(matrix, vector)
```

In [ ]:
```
print("Output2:", output2)
```

```
Output2: [ 4.  8. -3.]
```

Part (d) -- 1pt

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

In [ ]:
```
#I would assume this question ask one to make a test case. So here's a simpl

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

```
The 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 [ ]:
```python
import time

def matrix_multiplication_manual(matrix, vector):
    output = [0] * len(matrix)

    #Perform matrix multiplications
    for i in range(len(matrix)):
        sum = 0
        for j in range(len(vector)):
          sum += matrix[i][j] * vector[j]
    output[i] = sum

    return np.array(output)


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

# place code to run here
for i in range(10000):
    matrix_multiplication_manual(matrix, vector)

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

# compute the difference
diff1 = end_time - start_time


#Repeat for the numpy.dot method
start_time = time.time()
for i in range(10000):
    np.dot(matrix, vector)
end_time = time.time()
diff2 = end_time - start_time

print(f"Time taken by manual multiplication method: {diff1: .6f} seconds") #
print(f"Time taken by numpy.dot method: {diff2: .6f} seconds") #0.0160 secor
```

```
Time taken by manual multiplication method:  0.084137 seconds
Time taken by numpy.dot method:  0.009857 seconds
```

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 [ ]:
```
import matplotlib.pyplot as plt
import PIL
import urllib
import requests
from io import BytesIO
```

Part (a) -- 1 pt

This is a photograph of a dog whose name is Mochi.

 alt text

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.

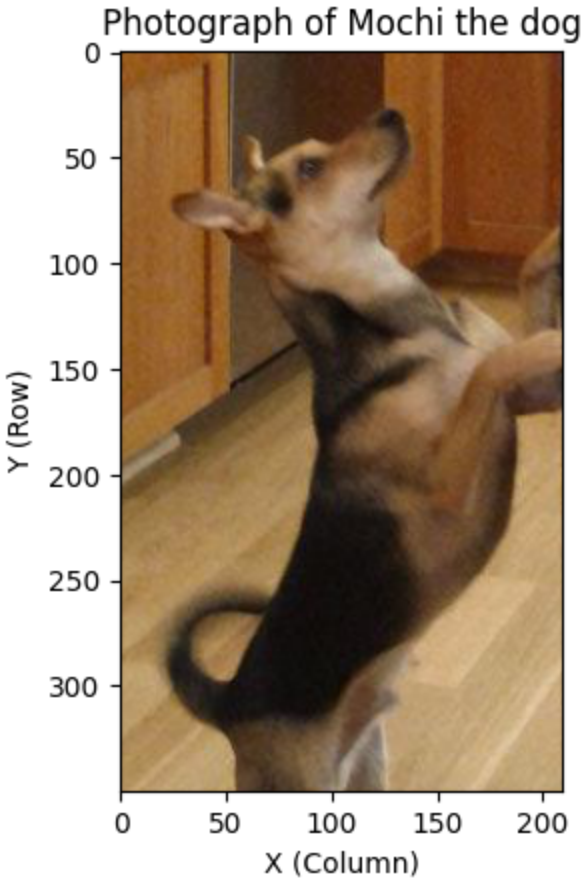
In [ ]:
```
# img = plt.imread("https://drive.google.com/uc?export=view&id=1oaLVR2hr1_qz
url = "https://drive.google.com/uc?export=view&id=1oaLVR2hr1_qzpKQ47i9rVUIkl
img = PIL.Image.open(urllib.request.urlopen(url))
```

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 [ ]:
```
plt.imshow(img)
plt.title("Photograph of Mochi the dog")
plt.xlabel("X (Column)")
plt.ylabel("Y (Row)")
plt.show()
```

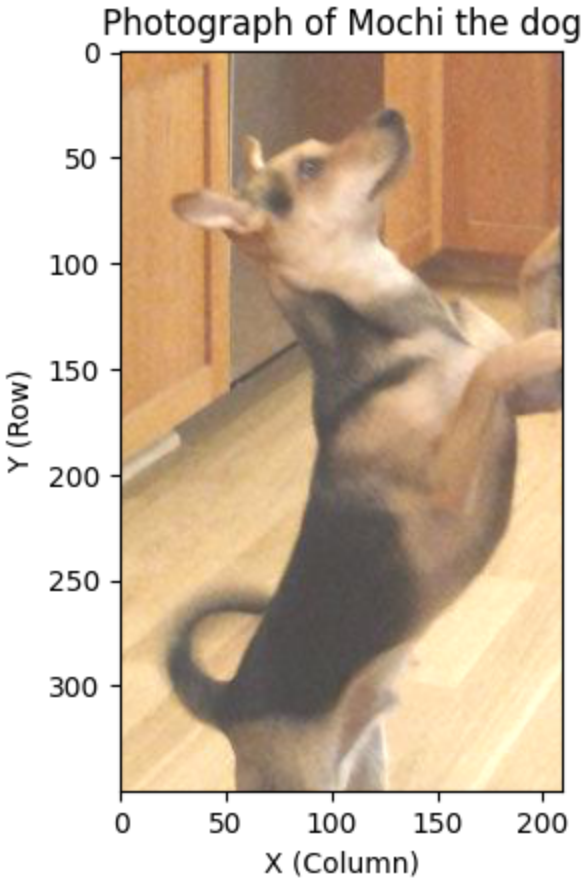


Part (c) -- 2pt

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 [ ]:
```
img_np = np.array(img) / 255.0
img_add = img_np + 0.25
img_add_clipped = np.clip(img_add, 0, 1)

plt.imshow(img_add_clipped)
plt.title("Photograph of Mochi the dog")
plt.xlabel("X (Column)")
plt.ylabel("Y (Row)")
plt.show()
```



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 [ ]:
```
img_cropped = img.crop((25, 25, 155, 175))  # (left, top, right, bottom)

# Convert the cropped image to RGB (discard alpha channel)
img_cropped = img_cropped.convert("RGB")

plt.imshow(img_cropped)
plt.title("Cropped Image of Mochi's Face")
plt.axis('off')  # Hide axes
plt.show()
```

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 [ ]:
```
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 [ ]:
```
img_torch = torch.from_numpy(np.array(img_cropped))
```

Part (b) -- 1pt

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

In [ ]:
```
print("Shape of img_torch: ", img_torch.shape)
```

```
Shape of img_torch:  torch.Size([150, 130, 3])
```

Part (c) -- 1pt

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

In [ ]:
```
num_floats = img_torch.numel()
print("Number of floating-point numbers in img_torch:", num_floats)
#basically 150 * 130 * 3 = 58,500
```

```
Number of floating-point numbers in img_torch: 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 [ ]:
```
"""Answer...
The code `img_torch.transpose(0, 2)` rearranges the dimensions of the `img_t
In this case, it swaps the first and third dimensions of the tensor while ke
dimension unchanged. This means that the order of how the tensor's data is c
these dimensions is changed, but the actual data in the tensor remains the s

The original variable img_torch is not updated because PyTorch tensor operat
do not modify tensors in place unless specifically indicated. Instead, they
with the operation applied."""


# Example tensor with shape [3, 4, 5]
tensor = torch.randn(3, 4, 5)
transposed_tensor = tensor.transpose(0, 2)


print("Original shape:", tensor.shape)
print("Transposed shape:", transposed_tensor.shape)
```

```
Original shape: torch.Size([3, 4, 5])
Transposed shape: torch.Size([5, 4, 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 [ ]:
```
"""Answer...
The code img_torch.unsqueeze(0) adds a new dimension (of size 1) at the
specified position (0 in this case) to the PyTorch tensor img_torch.
```

```
This operation does not modify the original tensor but instead
returns a new tensor with the additional dimension."""

# Example tensor with shape [3, 4, 5]
tensor = torch.randn(3, 4, 5)
new_tensor = tensor.unsqueeze(0)

print("Original shape:", tensor.shape)
print("Shape after unsqueeze(0):", new_tensor.shape)
```

```
Original shape: torch.Size([3, 4, 5])
Shape after unsqueeze(0): torch.Size([1, 3, 4, 5])
```

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 [ ]:
```
max_values, _ = torch.max(img_torch, dim=0)
print("Maximum values along each color channel:\n", max_values)
```

```
Maximum values along each color channel:
 tensor([[171, 118,  66],
        [177, 123,  66],
        [169, 120,  65],
        [170, 120,  67],
        [171, 121,  76],
        [172, 118,  74],
        [171, 121,  83],
        [171, 124,  74],
        [171, 120,  84],
        [171, 122,  93],
        [172, 125,  96],
        [170, 124,  94],
        [169, 127,  93],
        [172, 134,  98],
        [178, 140, 104],
        [181, 146, 118],
        [192, 153, 125],
        [198, 160, 132],
        [196, 159, 131],
        [191, 153, 124],
        [190, 153, 124],
        [195, 158, 131],
        [204, 167, 141],
        [204, 167, 140],
        [205, 165, 139],
        [201, 161, 135],
        [192, 151, 125],
        [182, 141, 113],
        [166, 125,  97],
        [164, 125,  94],
        [166, 127,  96],
        [167, 131,  98],
        [166, 128,  96],
        [161, 126,  94],
        [159, 124,  92],
        [154, 118,  86],
        [162, 123,  84],
        [169, 129,  88],
        [198, 157, 111],
        [217, 182, 140],
        [218, 188, 150],
        [219, 190, 156],
        [181, 155, 122],
        [150, 124,  89],
        [137, 109,  72],
        [140, 110,  73],
        [148, 116,  78],
        [162, 128,  86],
        [169, 133,  81],
        [176, 139,  87],
        [177, 140,  91],
        [185, 147,  98],
        [189, 148, 102],
        [182, 142, 102],
        [187, 149, 110],
```

```
[180, 145, 107],
[174, 139,  99],
[174, 137,  95],
[175, 137,  97],
[178, 136,  99],
[188, 146, 101],
[199, 154, 105],
[205, 159, 108],
[202, 155, 117],
[205, 158, 120],
[204, 157, 123],
[200, 154, 124],
[199, 156, 126],
[197, 158, 128],
[191, 156, 126],
[190, 157, 126],
[183, 148, 120],
[185, 151, 123],
[188, 155, 124],
[185, 155, 121],
[183, 157, 124],
[182, 157, 126],
[183, 158, 127],
[186, 158, 130],
[192, 159, 135],
[191, 158, 134],
[192, 159, 135],
[192, 162, 137],
[192, 163, 139],
[193, 163, 141],
[189, 162, 141],
[188, 161, 142],
[190, 163, 143],
[193, 167, 150],
[194, 170, 152],
[192, 170, 147],
[190, 168, 147],
[188, 170, 148],
[193, 175, 153],
[194, 176, 154],
[211, 185, 169],
[202, 191, 164],
[195, 183, 156],
[204, 190, 164],
[210, 191, 168],
[215, 189, 162],
[228, 201, 172],
[220, 191, 161],
[216, 189, 166],
[205, 188, 166],
[204, 186, 166],
[203, 186, 163],
[207, 193, 167],
[205, 190, 164],
[197, 184, 160],
[198, 193, 161],
```

```
        [193, 169, 145],
        [188, 163, 141],
        [181, 145, 123],
        [183, 145, 110],
        [184, 146, 110],
        [183, 145, 111],
        [182, 146, 112],
        [183, 149, 114],
        [181, 146, 114],
        [177, 140, 113],
        [176, 142, 113],
        [173, 146, 119],
        [172, 145, 118],
        [168, 141, 117],
        [168, 140, 117],
        [171, 142, 118],
        [163, 144, 117],
        [161, 139, 113],
        [169, 138, 110]], 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 [ ]:
```
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

iteration_list = [1, 5, 10]
hidden_units_list = [10, 30, 50]
learning_rate_list = [0.001, 0.005, 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, hidden_units_list[0])
        self.layer2 = nn.Linear(hidden_units_list[0], 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=learning_rate_list[0], momentu



for i in range (iteration_list[2]):
  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 e
      optimizer.step()                   # step 4 (make the updates for each
      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: 0.024
Training Accuracy: 0.976
Test Error Rate: 0.101
Test Accuracy: 0.899
```

Iterations	Hidden Units	Learning Rate	Training Error	Training Accuracy	Test Error	Test Accuracy	Notes
1	10	0.001	0.086	0.914	0.131	0.869	
1	10	0.005	0.047	0.953	0.104	0.896	*
1	10	0.01	0.057	0.943	0.116	0.884	
1	30	0.001	0.078	0.922	0.113	0.887	
1	30	0.005	0.036	0.964	0.079	0.921	*
1	30	0.01	0.039	0.961	0.082	0.918	
1	50	0.001	0.072	0.928	0.112	0.888	
1	50	0.005	0.033	0.967	0.74	0.926	*
1	50	0.01	0.034	0.966	0.087	0.913	
5	10	0.001	0.037	0.963	0.098	0.902	
5	10	0.005	0.02	0.98	0.085	0.915	
5	10	0.01	0.039	0.961	0.073	0.927	*odd?
5	30	0.001	0.023	0.977	0.082	0.918	
5	30	0.005	0.011	0.989	0.066	0.934	
5	30	0.01	0.014	0.986	0.065	0.935	*odd?
5	50	0.001	0.019	0.981	0.077	0.923	
5	50	0.005	0.002	0.998	0.066	0.934	*
5	50	0.01	0.021	0.979	0.074	0.926	
10	10	0.001	0.024	0.976	0.101	0.899	
10	10	0.005	0.006	0.994	0.076	0.924	*
10	10	0.01	0.017	0.983	0.095	0.905	
10	30	0.001	0.003	0.997	0.073	0.927	
10	30	0.005	0.001	0.999	0.059	0.941	
10	30	0.01	0.001	0.999	0.054	0.946	*highest (odd?)
10	50	0.001	0.003	0.997	0.073	0.927	
10	50	0.005	0	1	0.057	0.943	*
10	50	0.01	0.016	0.984	0.086	0.914	

Table above shows three options (iterations, hidden units, and learning rate) with three different parameters of each, showing how they affect the training and testing accuracy. The one with asterisk sign is the most accurate option out of three learning rates within each set. Note: the numbers chosen for the best data below (for parts a b and c) are only reasonable within this test.

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?

The first observable high accuracy on the training data was achieved by increasing the number of hidden units to 50, the iterations to 10, and setting the learning rate at 0.005, resulting in approximately 100% accuracy. Generally, it can be observed that increasing the number of hidden units and iterations with a moderate learning rate tends to improve training accuracy.

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 accuracy on the testing data with the highest learning rate at 0.01, combined with the highest iterations at 10 but surprisingly only at 30 hidden units, was achieved at 94.6% accuracy. This could be due to the higher learning rate enabling faster convergence during the initial stages of training, although it may also lead to overshooting the optimal parameters.

Part (c) -- 4 pt

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

Based on the observations from the training and testing accuracies, the model hyperparameters from (b) should be used. While the model in (a) achieved the highest training accuracy, the model in (b) showed the lowest error and highest testing accuracy. This suggests that the model in (b) generalizes better to unseen data, making it a more reliable choice for practical applications. Therefore, using a moderate number of hidden units (30), a relatively high learning rate (0.01), and an optimal number of iterations (10) should provide a balance between training and testing data but more priority in the ladder. (Note: the numbers chosen are only reasonable within this test).