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/1VUxOdSt4PZKkjkt-WVeEP7Wx2YzcsiON?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`.

```
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 and type(n) == int:
      for i in range (n + 1):
        sum = sum + (i ** 3)
      return sum
    else:
      print("Invalid input")
      return -1

print(sum_of_cubes(3))
```

```
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

```
help(str.split)

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]
    """
    word = sentence.split()
    length = []

    for i in word:
      length.append(len(i))

    return length

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 in the string is the same length. You should 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)

    for i in range (len(lengths) - 1):
      if lengths[i] != lengths [i + 1]:
        return False

    return True

all_same_length("all same length")
```

```
False
```

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`.

```
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.

```
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

# .size function represents the number of elements present in the matrix (12)
```

```
12
```

```
matrix.shape

# .shape function represents the dimensions of the matrix (rows(3), columns(4))
```

```
(3, 4)
```

```
vector.size

# .size function represents the number of elements present in the vector (4)
```

```
4
```

```
vector.shape

# .shape function represents the dimensions of the vector (rows(4), columns(1))
```

```
(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

```
output = None
```

```
output = np.zeros(matrix.shape[0])
temp = 0

for i in range(matrix.shape[0]):
  for j in range(matrix.shape[1]):
    temp = matrix [i, j] * vector[j]
    output[i] = output[i] + temp

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.

```
output2 = None

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.

```
print (output == output2)
```

```
[ True  True  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

# 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
```

```
0.00096893310546875
```

```
# initial time
start_time_loop = time.time()

# matrix multiplication using a for loop
output3 = np.zeros(matrix.shape[0])
Variable = 0

for i in range(matrix.shape[0]):
  for j in range(matrix.shape[1]):
    var = matrix [i, j] * vector[j]
    output3[i] = output3[i] + var

# end time
end_time_loop = time.time()

# difference
diff_loop = end_time_loop - start_time_loop

# initial time
start_time_np = time.time()

# matrix multiplication using .dot function
output4 = np.dot(matrix, vector)

# end time
end_time_np = time.time()

# difference
diff_np = end_time_np - start_time_np

# difference between loop and .dot function
difference = diff_np - diff_loop
difference

# because the difference between loop and np > 0, using .dot function
is faster
```

```
0.0008637905120849609
```

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.

```
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) 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.

```
import PIL, urllib
img =
np.array(PIL.Image.open(urllib.request.urlopen("https://drive.google.com/uc?export=view&id=1oaLVR2hr1_qzpKQ47i9rVUIklwbDcews")))/255
```

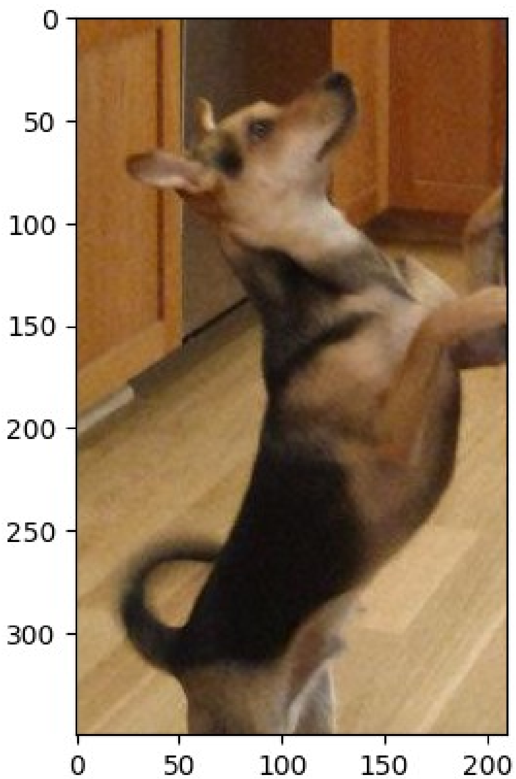
Part (b) -- 1pt

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.

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

```
<matplotlib.image.AxesImage at 0x7c84003b7580>
```

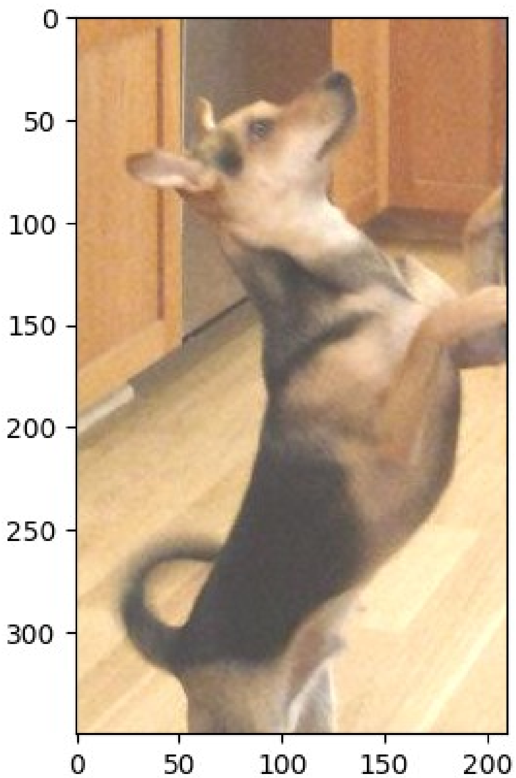


Part (c) -- 2pt

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`.

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

```
<matplotlib.image.AxesImage at 0x7c84002ee680>
```



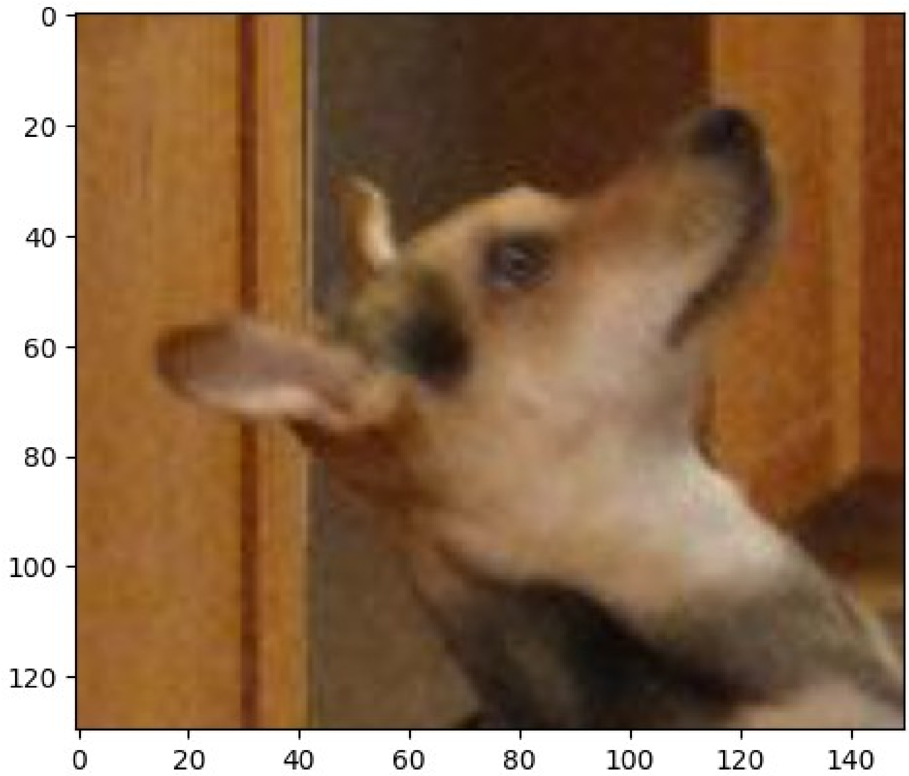
Part (d) -- 2pt

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.

```
img_cropped = img[10:140, 10:160, 0:3]
plt.imshow(img_cropped)
print(img_cropped.shape)
```

```
(130, 150, 3)
```



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.

```
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`.

```
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`?

```
img_torch.numel()
```

```
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.

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

# the transpose expression returns the matrix of img_torch after
switching the
# 0th and 2nd dimension

# this is confirmed by calling the .shape function which prints [3,
150, 130]

# the original variable img_torch is not updated because the
transposed matrix
# was never set to the original one
```

```
tensor([[[0.5529, 0.5529, 0.5451,  ..., 0.5922, 0.6000, 0.6039],
         [0.5569, 0.5647, 0.5608,  ..., 0.5961, 0.6000, 0.6000],
         [0.5725, 0.5804, 0.5725,  ..., 0.6000, 0.5961, 0.5922],
         ...,
         [0.5098, 0.5020, 0.4941,  ..., 0.5490, 0.5765, 0.6118],
         [0.5373, 0.5216, 0.4745,  ..., 0.5176, 0.5725, 0.6078],
         [0.5216, 0.5059, 0.4745,  ..., 0.4980, 0.5686, 0.6078]],

        [[0.3412, 0.3412, 0.3451,  ..., 0.3725, 0.3804, 0.3804],
         [0.3412, 0.3490, 0.3608,  ..., 0.3765, 0.3804, 0.3765],
         [0.3529, 0.3647, 0.3686,  ..., 0.3804, 0.3725, 0.3686],
         ...,
         [0.2431, 0.2353, 0.2431,  ..., 0.4745, 0.5059, 0.5412],
         [0.2627, 0.2471, 0.2235,  ..., 0.4392, 0.4941, 0.5294],
```

```
         [0.2471, 0.2314, 0.2235,  ..., 0.4118, 0.4824, 0.5176]],

        [[0.0588, 0.0588, 0.1216,  ..., 0.1804, 0.1961, 0.2039],
         [0.0784, 0.0863, 0.1451,  ..., 0.1922, 0.1961, 0.2000],
         [0.1137, 0.1137, 0.1725,  ..., 0.1961, 0.1961, 0.1922],
         ...,
         [0.0980, 0.0902, 0.0941,  ..., 0.4196, 0.4510, 0.4863],
         [0.1216, 0.1059, 0.0745,  ..., 0.3961, 0.4510, 0.4863],
         [0.1059, 0.0902, 0.0745,  ..., 0.3686, 0.4392, 0.4863]]],
       dtype=torch.float64)
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 original variable `img_torch` updated? Explain.

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

# the unsqueeze expression adds a dimension of 1 in the 1st position
of the
# matrix, resulting in a new shape of [1, 130, 150, 3]

# the original variable img_torch is not updated because the
unsqueezed matrix
# was never set to the original one

tensor([[[[0.5529, 0.3412, 0.0588],
          [0.5569, 0.3412, 0.0784],
          [0.5725, 0.3529, 0.1137],
          ...,
          [0.5098, 0.2431, 0.0980],
          [0.5373, 0.2627, 0.1216],
          [0.5216, 0.2471, 0.1059]],

         [[0.5529, 0.3412, 0.0588],
          [0.5647, 0.3490, 0.0863],
          [0.5804, 0.3647, 0.1137],
          ...,
          [0.5020, 0.2353, 0.0902],
          [0.5216, 0.2471, 0.1059],
          [0.5059, 0.2314, 0.0902]],

         [[0.5451, 0.3451, 0.1216],
          [0.5608, 0.3608, 0.1451],
          [0.5725, 0.3686, 0.1725],
          ...,
          [0.4941, 0.2431, 0.0941],
```

```
          [0.4745, 0.2235, 0.0745],
          [0.4745, 0.2235, 0.0745]],

         ...,

         [[0.5922, 0.3725, 0.1804],
          [0.5961, 0.3765, 0.1922],
          [0.6000, 0.3804, 0.1961],
          ...,
          [0.5490, 0.4745, 0.4196],
          [0.5176, 0.4392, 0.3961],
          [0.4980, 0.4118, 0.3686]],

         [[0.6000, 0.3804, 0.1961],
          [0.6000, 0.3804, 0.1961],
          [0.5961, 0.3725, 0.1961],
          ...,
          [0.5765, 0.5059, 0.4510],
          [0.5725, 0.4941, 0.4510],
          [0.5686, 0.4824, 0.4392]],

         [[0.6039, 0.3804, 0.2039],
          [0.6000, 0.3765, 0.2000],
          [0.5922, 0.3686, 0.1922],
          ...,
          [0.6118, 0.5412, 0.4863],
          [0.6078, 0.5294, 0.4863],
          [0.6078, 0.5176, 0.4863]]]], dtype=torch.float64)
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-dimensional PyTorch tensor with exactly three values.

Hint: lookup the function `torch.max`.

```
temp = torch.max(img_torch, 0)[0]
max_value = torch.max(temp, 0)
max_value[0]

tensor([0.8941, 0.7882, 0.6745], dtype=torch.float64)
```

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.

```
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" task
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 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: 0.036
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?

```
# changes that were made:

  # training iterations (initial value = 1)
  # 10 iterations = 0.999 training accuracy
  # 20 iterations = 0.999 training accuracy
  # 30 iterations = 1.0 training accuracy

  # number of hidden units (initial value = 30)
  # 50 hidden units = 0.967 training accuracy
  # 100 hidden units = 0.97 training accuracy
  # 200 hidden units = 0.972 training accuracy
```

```
  # learning rate (initial value = 0.005)
  # 0.01 rate = 0.961 training accuracy
  # 0.02 rate = 0.81 training accuracy
  # 0.001 rate = 0.922 training accuracy

# out of the changes that were made, increasing the number of
iterations
# resulted in an inrease in training accuracy

# at 30 iterations, a training accuracy of 1.0 (100%) was achieved
```

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?

```
# changes that were made:

  # training iterations (initial value = 1)
  # 10 iterations = 0.9410000000000001 testing accuracy
  # 20 iterations = 0.942 testing accuracy
  # 30 iterations = 0.9410000000000001 testing accuracy

  # number of hidden units (initial value = 30)
  # 50 hidden units = 0.926 testing accuracy
  # 100 hidden units = 0.923 testing accuracy
  # 200 hidden units = 0.927 testing accuracy

  # learning rate (initial value = 0.005)
  # 0.01 rate = 0.918 testing accuracy
  # 0.02 rate = 0.81 testing accuracy
  # 0.001 rate = 0.887 testing accuracy

# out of the changes that were made, changing the number of iterations
to 20
# resulted in an increase in testing accuracy

# at 20 iterations, a testing accuracy of 0.942 (94.2%) was achieved
```

Part (c) -- 4 pt

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

```
# you should use the hyperparameters from (b) because you want the
highest test
# accuracy

# test data is more important than training data
```