Lab Report

Homework 4

Name **Jing Bi**
(jb2548)
Xuanrui Zhang
(xz572)

Programming Exercises

1. **Approximating images with neural networks.**

a. Describe the structure of the network. How many layers does this network have? What is the purpose of each layer?

```
# layer_defs = [];
# layer_defs.push({type:'input', out_sx:1, out_sy:1, out_depth:2}); //
# layer_defs.push({type:'fc', num_neurons:20, activation:'relu'});
# layer_defs.push({type:'fc', num_neurons:20, activation:'relu'});
# layer_defs.push({type:'fc', num_neurons:20, activation:'relu'});
# layer_defs.push({type:'fc', num_neurons:20, activation:'relu'});
# layer_defs.push({type:'fc', num_neurons:20, activation:'relu'});
# layer_defs.push({type:'fc', num_neurons:20, activation:'relu'});
# layer_defs.push({type:'fc', num_neurons:20, activation:'relu'});
# layer_defs.push({type:'regression', num_neurons:3}); // 3 outputs: r,

# net = new convnetjs.Net();
# net.makeLayers(layer_defs);

# trainer = new convnetjs.SGDTrainer(net, {learning_rate:0.01, momentum
```

- There are 9 layers if including the input and regression layers.
- Among the layers, there are 7 fully connected hidden layers in this example according to the source code, each has type 'fc' and activation 'relu', and the number of neurons is 20.

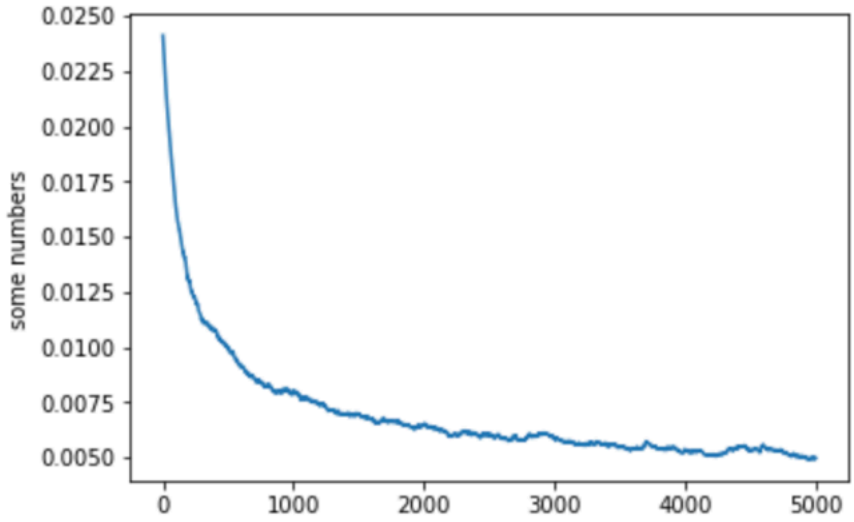
b. What does "Loss" mean here? What is the actual loss function? You may need to consult the source code, which is available on Github.

- The loss given in the example is actually a weighted sum of loss functions. Since this is a regression neural network, the actual (smooth) loss function is given in the source code:
- smooth_loss = 0.99 * smooth_loss + 0.01 * loss

```
import csv
losses = []
with open('LOSS.csv') as loss:
    csvReader = csv.reader(loss)
    for row in csvReader:
        losses.append(row)
```

```
loss = []
for i in range(len(losses)):
    if i == 0:
        cur = losses[i][0].strip('\ufeffVM94:1 ')
        loss.append(cur)
    elif i > 0:
        cur = losses[i][0].strip('VM94:1 ')
        loss.append(cur)
```

c. Plot the loss over time, after letting it run for 5,000 iterations. How good does the network eventually get?



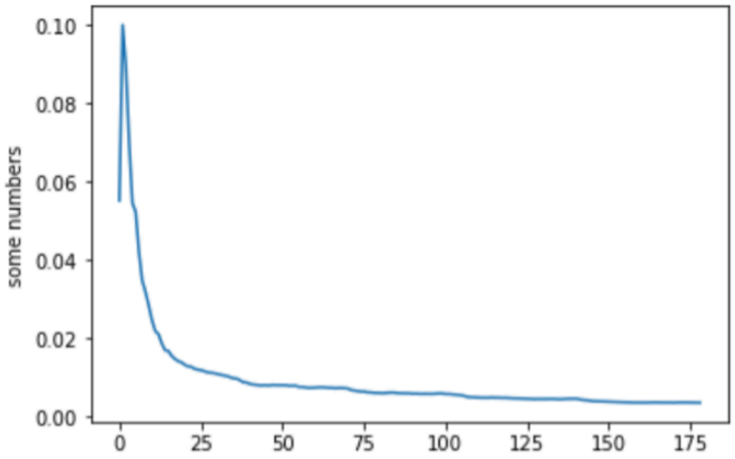

```
1 import matplotlib.pyplot as plt
2 x = list(range(5000))
3 plt.plot(loss)
4 plt.ylabel('some numbers')
5 plt.show()
```

```
1 print(loss[-1])
```

```
0.004970138822497
```

- The loss eventually gets down to 0.0049 at the iteration of 5,000.

d. Can you make the network converge to a lower loss function by lowering the learning rate every 1,000 iterations? (Some *learning rate schedules*, for example, halve the learning rate every *n* iterations. Does this technique let the network converge to a lower training loss?)



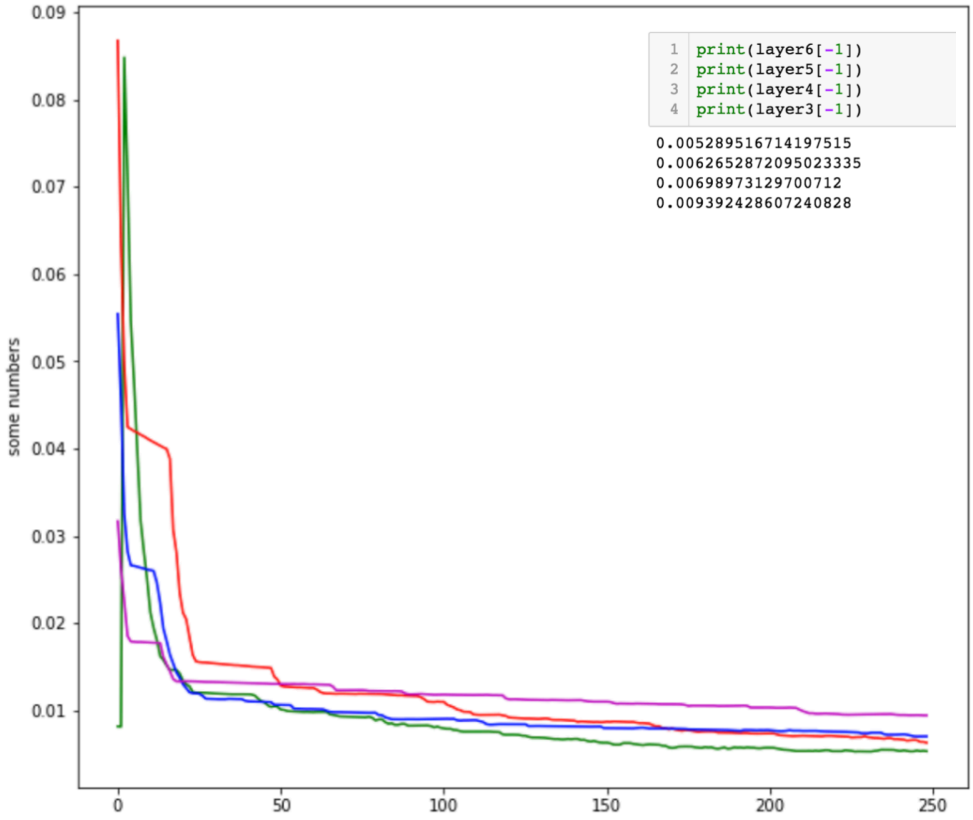

```
decrease = []
with open('Decrease_loss.csv') as loss:
    csvReader = csv.reader(loss)
    for row in csvReader:
        decrease.append(row)

decease_loss = []
for i in range(len(decrease)):
    if i == 0:
        cur = decrease[i][0].strip('\ufeffVM94:1 Loss:')
        decease_loss.append(float(cur))
    elif i > 0:
        cur = decrease[i][0].strip('VM26:1 Loss:')
        decease_loss.append(float(cur))
```

```
plt.plot(decease_loss)
plt.ylabel('some numbers')
plt.show()
print(decease_loss[-1])
```

```
0.003625026115127794
```

- Here we decrease the learning rate by approximately 0.02 every 1,000 iterations.
- To be more specific, at each 1,000 iterations, we decreased to 0.0079, 0.006m 0.0044, and 0.0022.
- By decreasing the learning rate, we find that at iteration 5,000, we decrease the loss to 0.0036, which lower than the loss that we got before.

e. Lesion study. The text box contains a small snippet of Javascript code that initializes the network. You can change the network structure by clicking the "Reload network" button, which simply evaluates the code. Let's perform some brain surgery: Try commenting out each layer, one by one. Report some results: How many layers can you drop before the accuracy drops below a useful value? How few hidden units can you get away with before quality drops noticeably?



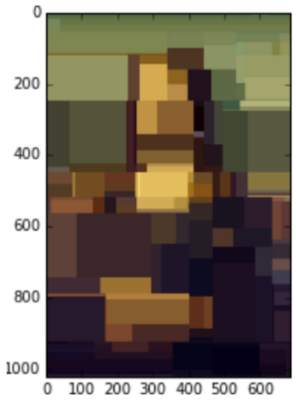
- If we drop the last 2 layers, we get loss value at around 0.0052 at the 5,000th iteration.
  If 3 layers are dropped, we will get 0.0055 at the 5,000th iteration.
  If 4 layers are dropped, the loss value will exceed 0.006 at the 5,000th iteration.
- To be more extreme, if there is only one hidden layer left, the loss value will increase to 0.026 at the 5,000th iteration and seems to say at around 0.0025-0,0026 regardless how many iterations are ran.
- Comparing to have 6, 5, 4, and 3 layers left, we can see that there is a huge gap between the 4th layer (loss = 0.0069) and the 3rd layer (loss = 0.009). Hence, we can drop 3 layers before the accuracy drops below an useful value, which means to drop 3*20 = 60 neurons.

f. Try adding a few layers by copy+pasting lines in the network definition. Can you noticeably increase the accuracy of the network?

- By adding more layers, we constantly get loss of value of 0.0058, no matter how many more layers we added. Hence, adding more layers will not necessarily lower the loss, which means that we will not be able to increase the accuracy of the network.
- As Professor Serge pointed out, adding more layers will add more parameters, which is why the loss will not decrease.

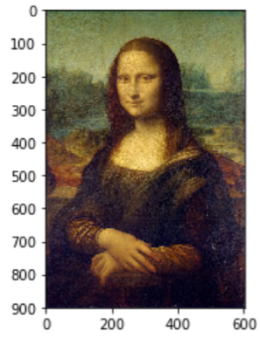
2. **Random forests for image approximation** In this question, you will use random forest regression to approximate an image by learning a function, $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, that takes *image* (x, y) *coordinates* as input and outputs *pixel brightness.* This way, the function learns to approximate areas of the image that it has not seen before.

a. Start with an image of the Mona Lisa. If you don't like the Mona Lisa, pick another interesting image of your choice.



b. Preprocessing the input. To build your "training set," uniformly sample 5,000 random (x, y) coordinate locations.

```
# load image
img = Image.open("monalisa")
img = np.asarray(img)
plt.imshow(img)
plt.show()
```



- Create 5,000 random locations

```
sample = 5000 #number of training points

#create sample points
data = []
label = []
row = img.shape[0]
col = img.shape[1]

for i in range(sample):

    r = rd.uniform()*row
    c = rd.uniform()*col
    point = [int(r), int(c)]

    data.append(point)
    label.append(img[point[0],point[1]])

data = np.asarray(data)
label = np.asarray(label)
```

- What other preprocessing steps are necessary for random forests inputs? Describe them, implement them, and justify your decisions. In particular, do you need to perform mean subtraction, standardization, or unit-normalization?

  - There is no other preprocessing steps;
  - In particular, we do not perform mean subtraction, standardization, or unit-normalization since decision tree will not be affected by the data structure.

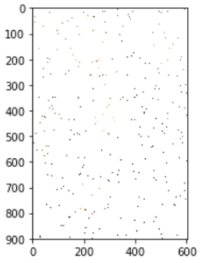
c. Preprocessing the output. Sample pixel values at each of the given coordinate locations. Each pixel contains red, green, and blue intensity values, so decide how you want to handle this. There are several options available to you:

- Convert the image to grayscale
- Regress all three values at once, so your function maps (x, y) coordinates to (r, g, b) values: $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3$
- Learn a different function for each channel, $f_{Red} : \mathbb{R}^2 \rightarrow \mathbb{R}$, and likewise for f_{Green}, f_{Blue}.

Note that you may need to rescale the pixel intensities to lie between 0.0 and 1.0. (The default for pixel values may be between 0 and 255, but your image library may have different defaults.)

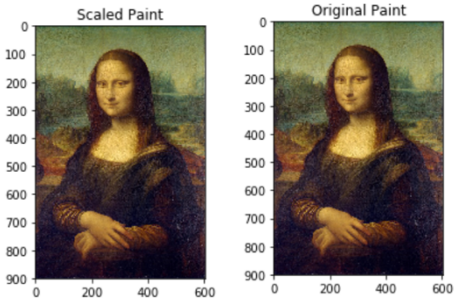
- **Perform the second option: regress all three values at once**

```
# create a white image
sample_img = np.zeros([row,col,img.shape[2]], dtype = np.uint8)
sample_img.fill(255)


for i in range(5000):
    idx = data[i][0]
    idy = data[i][1]
    sample_img[idx][idy] = label[i] #choose the selected sample points to show

plt.imshow(sample_img)
plt.show()
```



- **Rescale the pixel intensities**

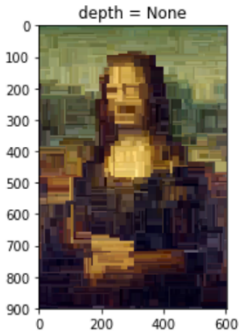
```
scaled = np.zeros(img.shape)

for i in range(row):
    for j in range(col):
        scaled[i,j] = img[i,j] / 255 #scale down from (0,255) to (0,1)

plt.title("Scaled Paint")
plt.imshow(scaled)
plt.show()

plt.title("Original Paint")
plt.imshow(img)
plt.show()

scale_label = label / 255 #apply same change to the label
```



What other preprocessing steps are necessary for random regression forest outputs? Describe them, implement them, and justify your decisions.

- **None**

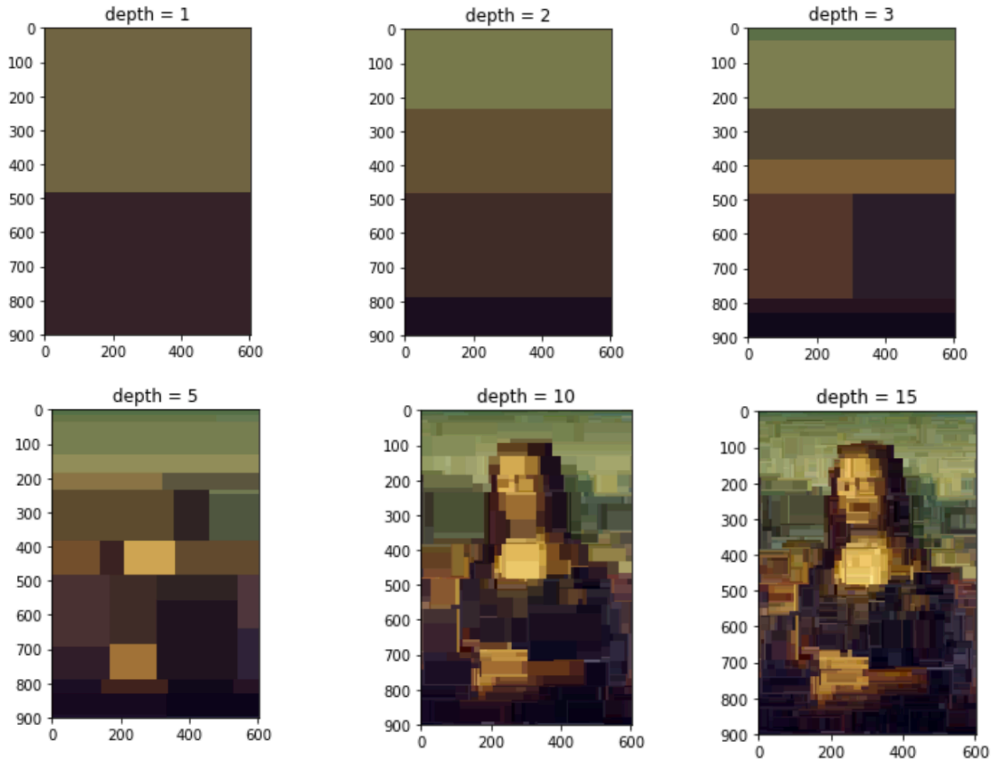
d. To build the final image, for each pixel of the output, feed the pixel coordinate through the random forest and color the resulting pixel with the output prediction. You can then use `imshow` to view the result. (If you are using grayscale, try `imshow(Y, cmap='gray')` to avoid fake-coloring). You may use any implementation of random forests, but you should understand the implementation and you must cite your sources.




```
def random_forest(dt, lab, n, depth):
    #to create and fit into the model
    rf = RandomForestRegressor(random_state=0, n_estimators = n, max_depth = depth)
    rf.fit(dt, lab)

    pred = np.zeros([row,col,3])

    for i in range(row):
        for j in range(col):
            point = np.array([i,j])
            pred[i,j] = rf.predict(point.reshape(1,-1)) / 255

    title = 'depth = ' + str(depth)
    plt.title(title)

    plt.imshow(pred)
    plt.show()

    return pred
```

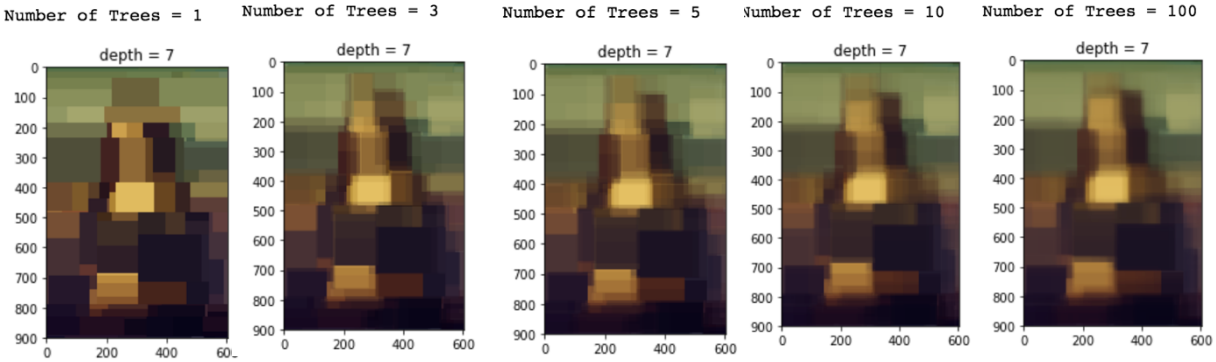
e. Experimentation.

i. Repeat the experiment for a random forest containing a single decision tree, but with depths 1, 2, 3, 5, 10, and 15. How does depth impact the result? Describe in detail why.



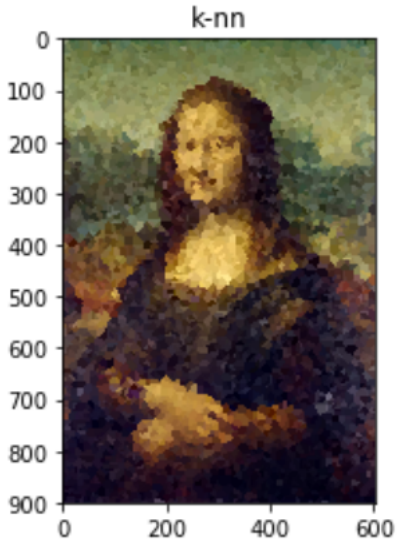
- Starting from depth = 1, the color of the image becomes binary that there are only 2 distinct colors in the graph. But as the depth increases, the number of color in the image increases as well with the number of 2^n, where depth = n. As the depth increases, the image becomes more and more precise and close to the original graph.

ii. Repeat the experiment for a random forest of depth 7, but with number of trees equal to 1, 3, 5, 10, and 100. How does the number of trees impact the result? Describe in detail why.



- As each tree comes with a different data set, it becomes smoother when we increase the number of trees. Since each set of data will result in highly correlated predictors that can limit the reduction of variance.

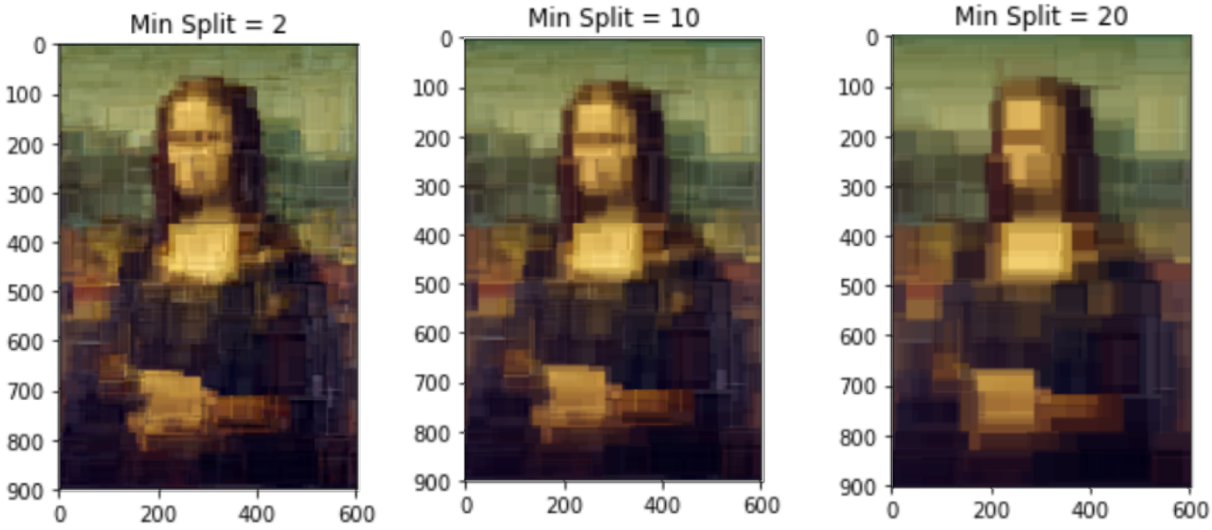
iii. As a simple baseline, repeat the experiment using a k-NN regressor, for $k = 1$. This means that every pixel in the output will equal the nearest pixel from the "training set." Compare and contrast the outlook: why does this look the way it does?



- In Random Forest, the image is split by comparing to the threshold of each subtree in the forest based on their pixel location. In result, the image will be constructed by small rectangles of same colors.
- In k-NN, the color of each pixel is decided by the closest one in the sample. In result, the image will be constructed by small pieces of colors with nonlinear boundaries.

iv. Experiment with different pruning strategies of your choice.

- We pruned with "min_samples_split" strategy where the values equal 2, 10, 20, respectively.



- As we can see from the above graphs, the images grow unclear with the number of splits increases.

f. Analysis.

i. What is the decision rule at each split point? Write down the 1-line formula for the split point at the root node for one the trained decision trees inside the forest. Feel free to define any variables you need.

- Input: the position of each pixel (x, y)
- Output: a node on the next level of the tree

- If we reach the leaf node, the output will be the color of the pixel, and the formula is:

```
if x >= threshold:
    next_step = left_tree
else:
    next_step = right_tree
```

ii. Why does the resulting image look like the way it does? What shape are the patches of color, and how are they arranged?

Random Forest:

- Since there are limited colors in our sample and in random forest, each part of the images will match with colors, the resulting image looks like the patches of color.
- The shape of patches is rectangle and they are arranged with overlapping pattern.

k-NN:

- Each pixel color is assigned to the closest one in the sample, thus the image is constructed by the small pieces of color while the boundaries of each piece are not straight lines.

iii. *Straightforward:* How many patches of color may be in the resulting image if the forest contains a single decision tree? Define any variables you need.

- Let depth = n, then the number of patches is: 2^n.
  Since each leaf will result in a patch of color, and we have a single complete binary tree in the graph. The number of patches equals to the number of leaves in the tree, which is 2^n.

iv. *Tricky:* How many patches of color might be in the resulting image if the forest contains *n* decision trees? Define any variables you need.

- Let depth = m, then the number of patches for each tree is 2^m. Since we have n decision trees, the number of patches will be $(n*2^m)$ choose n, which can be written as $C(n, n*2^m)$.

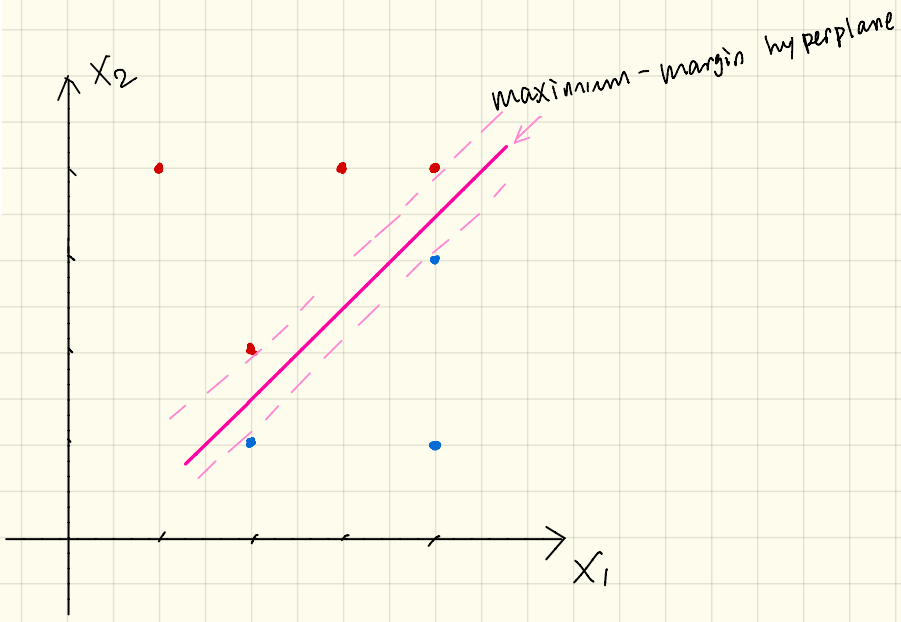
Written Exercise

1. **Maximum-margin classifiers** Suppose we are given $n = 7$ observations in $p = 2$ dimensions. For each observation, there is an associated class label.

1.(a)

a. Sketch the observations and the maximum-margin separating hyperplane.

X_1	X_2	Y
3	4	Red
2	2	Red
4	4	Red
1	4	Red
2	1	Blue
4	3	Blue
4	1	Blue



1.(b)

b. Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$, and classify to Blue otherwise." Provide the values for β_0, β_1, and β_2.

classify to Red if $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0$

classify to Blue if $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \leq 0$

$$\left.\begin{array}{l} \beta_0 + \beta_1(2) + \beta_2(2) > 0 \\ \beta_0 + \beta_1(4) + \beta_2(4) > 0 \\ \beta_0 + \beta_1(3) + \beta_2(4) > 0 \\ \beta_0 + \beta_1(1) + \beta_2(4) > 0 \\ \beta_0 + \beta_1(2) + \beta_2(1) \leq 0 \\ \beta_0 + \beta_1(4) + \beta_2(3) \leq 0 \\ \beta_0 + \beta_1(4) + \beta_2(3) \leq 0 \end{array}\right\} \Rightarrow$$

$\beta_0 + \beta_1(4) + \beta_2(3.5) = 0$

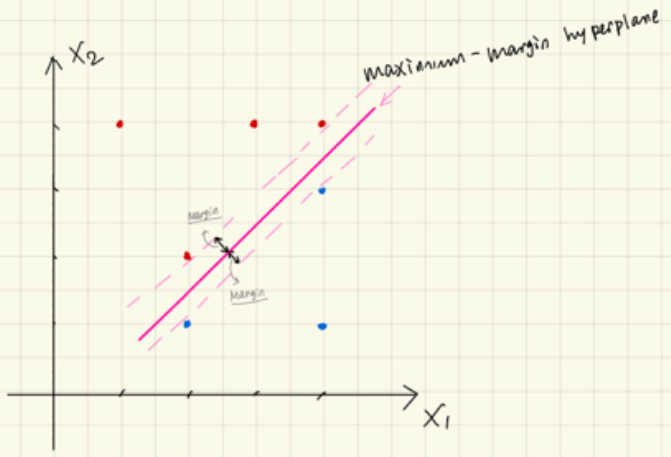
$\beta_0 > 0$, $\beta_1 < -\frac{5a}{6}$.

$\beta_2 = -\frac{2}{7}(\beta_0 + 4\beta_1)$

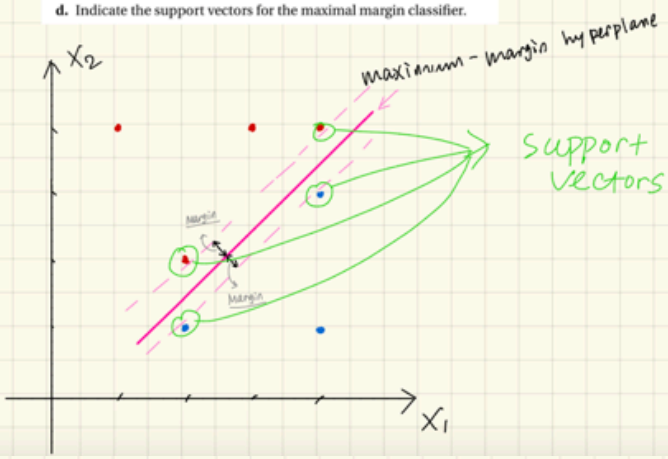
$\Rightarrow$ Choose $\beta_0 = 1$

$\beta_1 = -1$. $\beta_2 = \frac{6}{7}$

1.c **c.** On your sketch, indicate the margin for the maximal margin hyperplane.



1.d **d.** Indicate the support vectors for the maximal margin classifier.



1.e **e.** Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

$$f(X) = 1 - 1(X_1) + \frac{6}{7}(X_2)$$

seventh observation: $X_1 = 4$, $X_2 = 1$

$$f(x^7) = 1 - 4 + \frac{6}{7} < 0$$

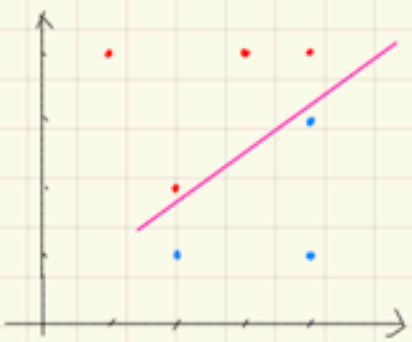
If we increase X_1^7 slightly., $f(x^7)$ ges less Since coefficient is negative

Similar to X_2^7. if we increase X_1^7 slightly, $f(x^7)$ still could be negative

Hence by slightly changing values of X^7, as long as 7th observation dont move across the margin line.

1.f

f. Sketch a hyperplane that separates the data, but is not the maximum-margin separating hyperplane. Provide the equation for this hyperplane.



$\beta_0 + \beta_1(2) + \beta_2(2) > 0$
$\beta_0 + \beta_1(4) + \beta_2(4) > 0$
$\beta_0 + \beta_1(3) + \beta_2(4) > 0$
$\beta_0 + \beta_1(1) + \beta_2(4) > 0$
$\beta_0 + \beta_1(2) + \beta_2(1) \leq 0$
$\beta_0 + \beta_1(4) + \beta_2(3) \leq 0$
$\beta_0 + \beta_1(4) + \beta_2(3) \leq 0$

$\beta_0 > 0$,
$-a < \beta_1 < -\frac{3a}{4}$
$\frac{1}{4}(-\beta_0 - 4\beta_1) < \beta_2 \leq -\beta_0 - 2\beta_1$

$\Rightarrow \beta_0 = 1$,
$\beta_1 = -\frac{7}{8}$,
$\beta_2 = \frac{6}{8}$

$$f(x) = 1 - \frac{7}{8}x_1 + \frac{6}{8}x_2$$

1.e

e. Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

$$f(x) = 1 - 1(x_1) + \frac{6}{7}(x_2)$$

seventh observation: $x_1 = 4$, $x_2 = 1$

$$f(x^7) = 1 - 4 + \frac{6}{7} < 0$$

If we increase x_1^7 slightly., $f(x^7)$ ges less. Since coefficient is negative

Similar to x_2^7. if we increase x_1^7 slightly, $f(x^7)$ still could be negative

Hence by slightly changing values of x^7, as long as 7th observation don't move across the margin line.

2. **Neural networks as function approximators.** Design a feed-forward neural network to approximate the 1-dimensional function given in Fig. 1. The output should match exactly. How many hidden layers do you need? How many units are there within each layer? Show the hidden layers, units, connections, weights, and biases.

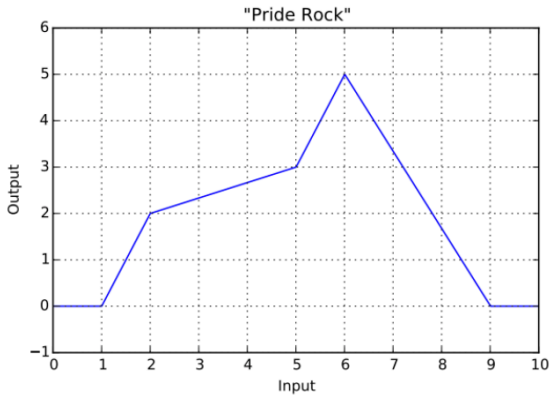


Figure 1: Example function to approximate using a neural network.

Use the ReLU nonlinearity for every unit. Every possible path from input to output must pass through the same number of layers. This means each layer should have the form

$$Y_i = \sigma(W_i Y_{i-1}^T + \beta_i), \tag{1}$$

where $Y_i \in \mathbb{R}^{d_i \times 1}$ is the output of the ith layer, $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$ is the weight matrix for that layer, $Y_0 = x \in \mathbb{R}^{1 \times 1}$, and the ReLU nonlinearity is defined as

$$\sigma(x) \triangleq \begin{cases} x & x \geq 0, \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

To write the functions shown in the graph as:

$$f(x) = \begin{cases} 0, & x \in [0, 1) \\ 2x-2, & x \in [1, 2) \\ \frac{1}{3}x + \frac{4}{3}, & x \in [2, 5) \\ 2x-7, & x \in [5, 6) \\ -\frac{5}{3}x+15, & x \in [6, 9) \\ 0, & x \in [9, 10) \end{cases}$$

Since the ReLU activition can be given by

$$\sigma(x) = \begin{cases} x, & x \geq 0 \\ 0, & \text{o/w} \end{cases}$$

Thus we write the function as:

$$f(x) = \begin{cases} 6(2x-2) & = 2x-2 \\ -6\left(\frac{5}{3}x - \frac{10}{3}\right) & = \frac{1}{3}x + \frac{4}{3} = (2x-2) - \left(\frac{5}{3}x - \frac{10}{3}\right) \\ +6\left(\frac{5}{3}x - \frac{25}{3}\right) & = 2x-7 = \left(\frac{1}{3}x + \frac{4}{3}\right) + \left(\frac{5}{3}x - \frac{25}{3}\right) \\ -6\left(\frac{11}{3}x - 22\right) & = -\frac{5}{3}x + 15 = (2x-7) - \left(\frac{11}{3}x - 22\right) \\ +6\left(\frac{5}{3}x - 15\right) & = 0 = \left(-\frac{5}{3}x + 15\right) + \left(\frac{5}{3}x - 15\right) \end{cases}$$

Thus, we get $W_1 = \left[2, \frac{5}{3}, \frac{5}{3}, \frac{11}{3}, \frac{5}{3}\right]$

$$W_{bias} = \left[-2, -\frac{10}{3}, -\frac{25}{3}, -22, -15\right]$$

and $W_2 = [1, -1, 1, -1, 1]$

The neural network has 1 hidden layer with 5 units.