CS 5785 Applied Machine learning

Homework 4

Team Member: Xueqi Wei, Queenie Liu

HW4_Q1

December 3, 2019

1 1. Approximating images with neural networks.

[1]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

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

The network has 9 layers, which includes 1 input layer, 1 output layer and 7 connected layers. The input layer declares 2-dimensional input points, (x,y) from 1 * 1 RGB image. The connected layer create layers of 20 neurons that use rectified linear units activation function, $f(x) = max(0, x)$. The output layer uses a regression layer to 3 real-valued output (r,g,b) to predict the color.

(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.

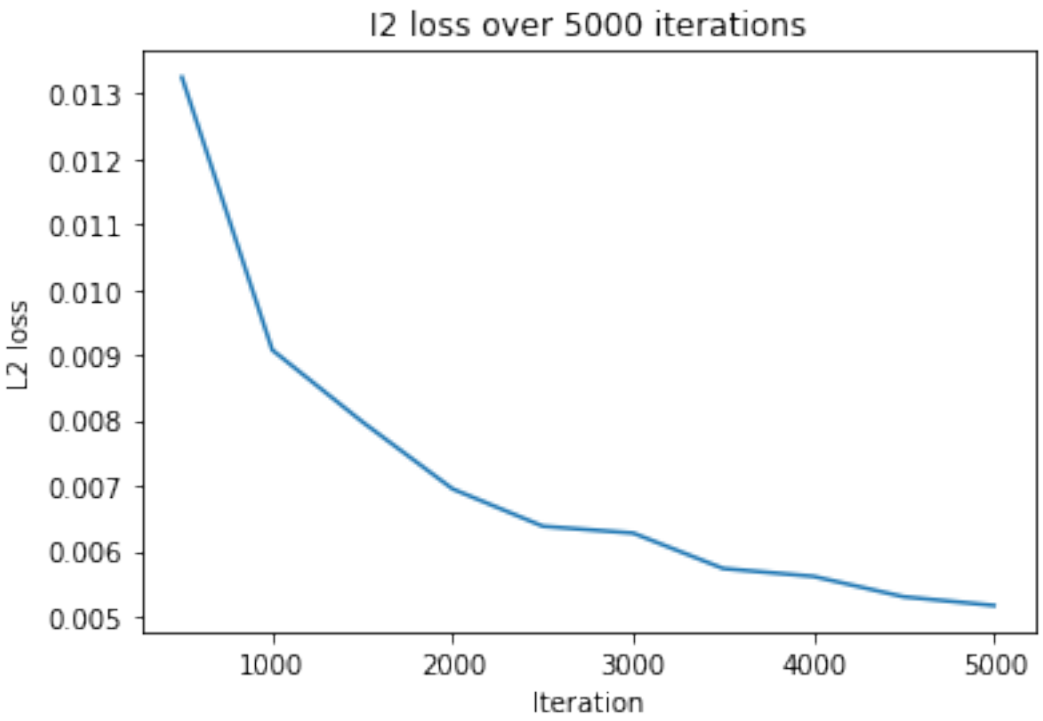
https://cs.stanford.edu/people/karpathy/convnetjs/docs.html

The "Loss" means L2 loss accured during backpropgation, which is:

$$L_i = f - y_{i2}^2$$

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

[18]:
```
#plot the loss over time with an interval of 500 iterations within 5000␣
↪iterations.
iteration = [498, 999, 1498, 1999, 2499, 2999, 3498, 3999, 4498, 4999]
l2_loss = [0.013251, 0.009076, 0.0079768, 0.0069489, 0.00637867, 0.00627410, 0.
↪0057321, 0.00561118, 0.0052997, 0.00516629]
plt.plot(iteration, l2_loss)
plt.title('I2 loss over 5000 iterations')
plt.xlabel('Iteration')
plt.ylabel('L2 loss')
plt.show()
```

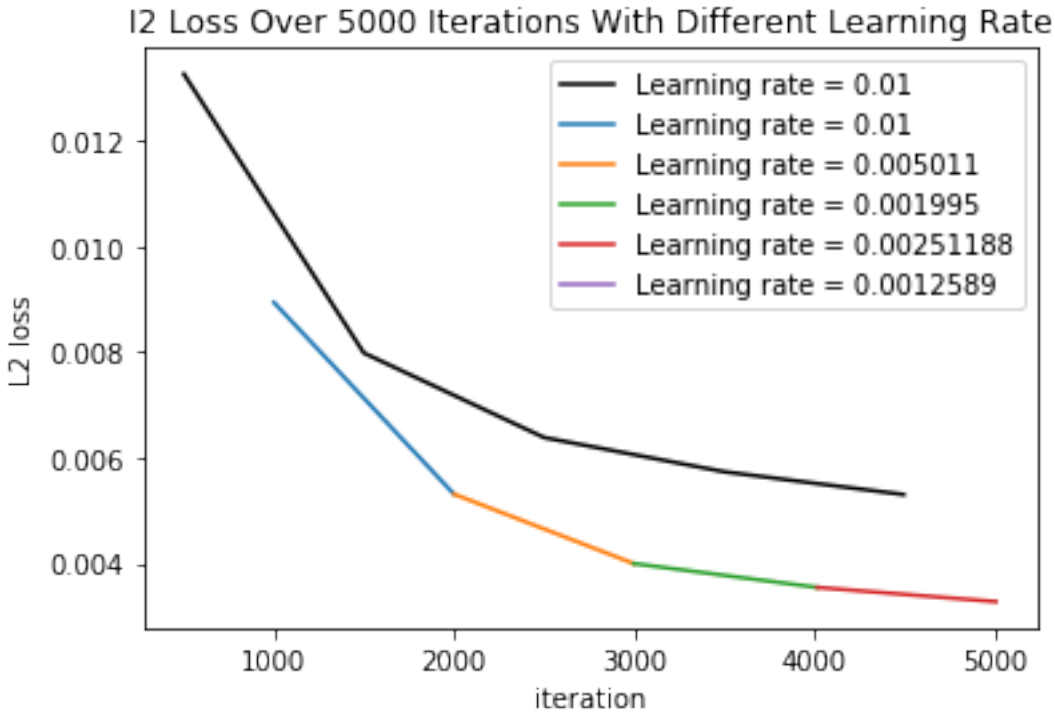


L2 loss decreases as iteration increases. The l2 loss decrease to around 0.005 when approaching 5000 iterations.

(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?)

[17]:
```
#plot the loss function by lowering the learning rate every 1000 iterations.
#plot the loss function without changing the learning rate every 1000␣
 ↪iterations.
#compare the two lines in one graph to see the changes.
iteration_2 = [999, 1999, 2998, 4017, 5011]
l2_loss_2 = [0.008931, 0.005308, 0.00399489, 0.00354431, 0.00327475]
rates = [0.01, 0.005011, 0.001995, 0.00251188, 0.0012589, 0.00050118]
plt.plot([iteration[2 * k] for k in range(0,5)], [l2_loss[2 * k] for k in␣
 ↪range(0,5)], 'k', label = 'Learning rate = 0.01')
plt.plot(iteration_2[0:2],l2_loss_2[0:2],label = ('Learning rate = 0.01'))
plt.plot(iteration_2[1:3],l2_loss_2[1:3], label = ('Learning rate = 0.005011'))
plt.plot(iteration_2[2:4],l2_loss_2[2:4], label = ('Learning rate = 0.001995'))
plt.plot(iteration_2[3:5],l2_loss_2[3:5], label = ('Learning rate = 0.
 ↪00251188'))
plt.plot(iteration_2[4:6],l2_loss_2[4:6], label = ('Learning rate = 0.0012589'))
plt.legend(loc='best')
plt.xlabel('Iteration')
```

```
plt.ylabel('L2 loss')
plt.title('I2 Loss Over 5000 Iterations With Different Learning Rate')
plt.show()
```



Lowering learning rate during iterations can help network coverage to a lower loss function.

(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?

Fix learning rate = 0.01

Drop 1, loss = 0.0055. Drop 2, loss = 0.0066. Drop 3 loss = 0.0071 Drop 4, loss = 0.0075, Drop 5, loss = 0.01. You can drop 2 layers before the accuracy drops below a useful value. You can drop 4 hidden units before quality drops noticeably.

Accuracy drops noticeably when number of hidden layers are less than 3.

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

After adding 4 layers, you cannot noticeably increase the accuracy of the network.

HW4_Q2

December 4, 2019

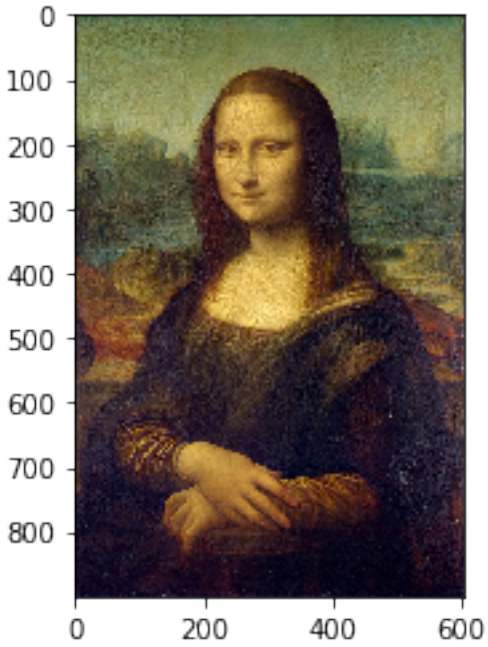
1 2.Random forests for image approximation

[14]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from PIL import Image
from sklearn.ensemble import RandomForestRegressor as rforest
from sklearn.neighbors import KNeighborsRegressor as knn
```

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

Figure 2: Left: http://tinyurl.com/mona-lisa-small Mona Lisa, Leonardo da Vinci, via Wikipedia. Licensed under Public Domain. Middle: Example output of a decision tree regressor. The input is a "feature vector" containing the (x, y) coordinates of the pixel. The output at each point is an (r, g ,b) tuple. This tree has a depth of 7. Right: Example output of a k-NN regressor, where k = 1. The output at each pixel is equal to its closest sample from the training set.

[15]:
```
im = Image.open("Mona_Lisa.jpg")
img = np.asarray(im)
w, h = im.size
#print(w, h)
#print(img.shape)
plt.imshow(im)
plt.show()
```



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

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?

[16]:
```
x = np.random.choice (w, 5000)
y = np.random.choice (h, 5000)
```

We do not need to perform mean subtraction, standardization, or unit-normalization. No other step is needed.

(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:

[17]:
```
sample_size = 5000
pix = [np.array(img[y[i], x[i]])/255 for i in range(sample_size)]
pix = np.array(pix)
data = np.array([y,x])
data = data.T
#pix
#data
```

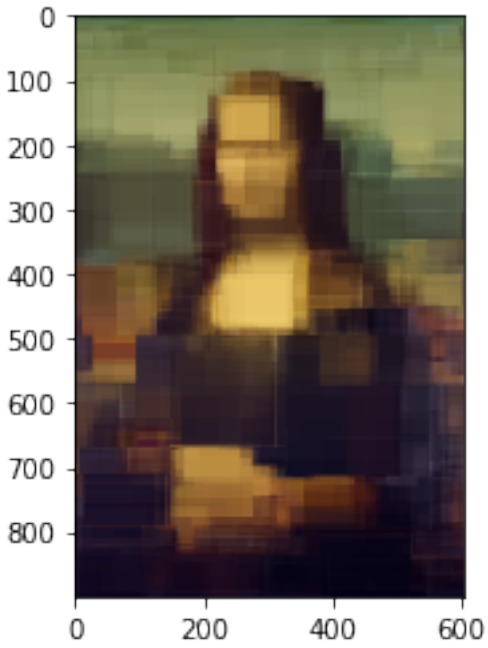
Make (x, y) into one sample.

(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.

[30]:
```
#from sklearn.ensemble import RandomForestRegressor as rforest
test =  [[i, j] for i in range(h) for j in range(w)]

regression = rforest(max_depth = 10 , n_estimators = 10, random_state = 0)
regression.fit(data, pix)
result = regression.predict(test)
result = result.reshape(h, w, 3)
plt.imshow(result)
plt.show
```

[30]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



(e) Experimentation.

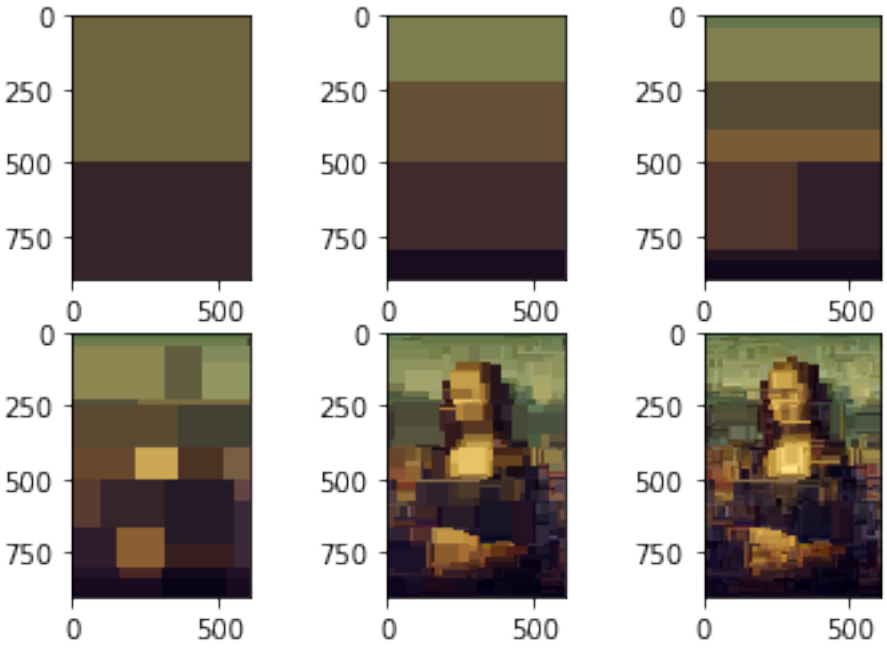
i. Repeat the experiment for a random forest containing a single decision tree, but with depths 1, 2, 3, 5, 10, and 15. How does depth impact the result? Describe in detail why.

[21]:
```
depth = [1, 2, 3, 5, 10, 15]
```

[22]:
```
#from sklearn.ensemble import RandomForestRegressor as rforest
for idx,i in enumerate(depth):

    regression = rforest(max_depth = i, n_estimators = 1, random_state = 0)
    regression.fit(data, pix)
    result = regression.predict(test)
    result = result.reshape(h, w, 3)

    plt.subplot(2, 3, idx + 1)
    plt.plot([0, 1], [0, idx + 1])
    plt.imshow(result)
```



When we increase the depth, the image become much clearer. Since the depth directly influence the average pixel size, the image gets better when the depth is increasing.

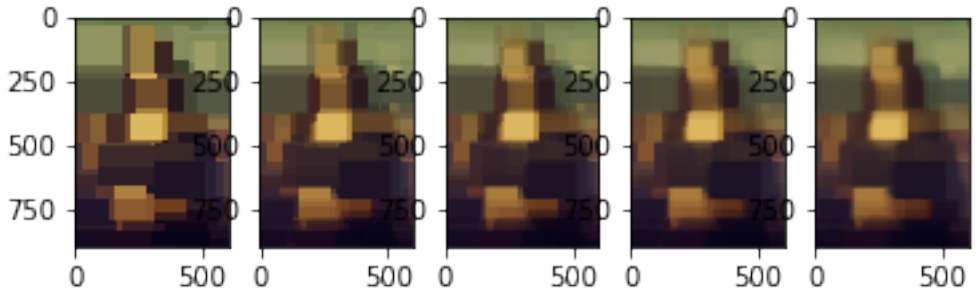
ii. Repeat the experiment for a random forest of depth 7, but with number of trees equal to 1, 3, 5, 10, and 100. How does the number of trees impact the result? Describe in detail why.

[24]:
```
tree = [1, 3, 5, 10, 100]
```

[31]:
```
for idx,i in enumerate(tree):

    regression = rforest(max_depth = 7, n_estimators = i, random_state = 0)
    regression.fit(data, pix)
    result = regression.predict(test)
    result = result.reshape(h, w, 3)
```

```
    plt.subplot(1, 5, idx + 1)
    plt.plot([0, 1], [0, idx + 1])
    plt.imshow(result)
```

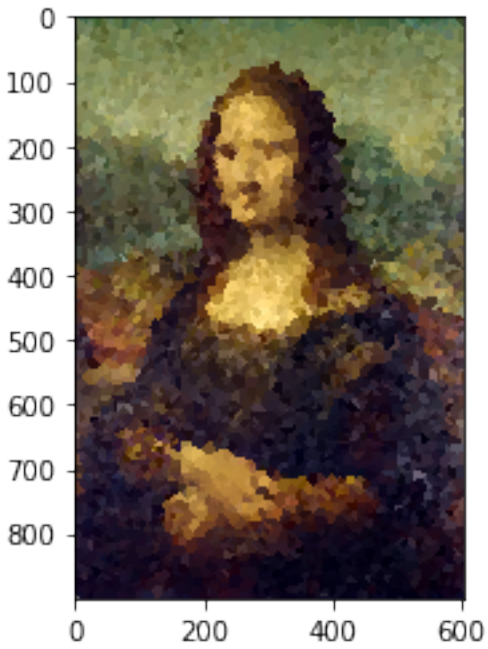


The pixel size is similar but the color contrast is getting more blended and smooth.This is because that the pixel colors are now chosen as a majority vote among the leaves so that increasing the number of trees will increase the color accuracy.

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?

[113]:
```
neighbors = knn (n_neighbors = 1)
neighbors.fit(data, pix)
test = [[i, j] for i in range (h) for j in range (w)]
out = neighbors.predict(test)
out = out.reshape(h, w, 3)
plt.imshow(out)
```

[113]:
```
<matplotlib.image.AxesImage at 0x1a1c5bb4e0>
```



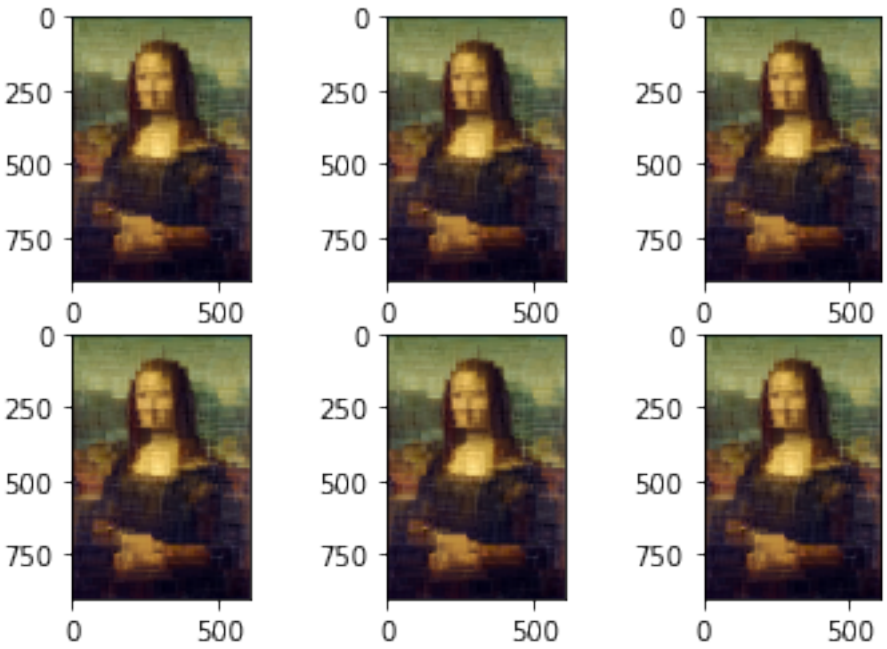
The result better resembles the original Mona Lisa image. Since we use the image without random sample, knn will help us make better approximation which better supports the way that the human eye processes images.

iv. Experiment with different pruning strategies of your choice.

[124]:
```
max_leaves = [5,10,20,40,100,150]
```

[125]:
```
for index,i in enumerate(max_leaves):
    plt.subplot(2,3,index+1)
    plt.plot([0,1],[0,index+1])

    max_leaves = rforest(max_depth = i, n_estimators = 10, random_state=0)
    max_leaves.fit(data, pix)
    out = max_leaf.predict(test)
    out = out.reshape(h, w, 3)
    plt.imshow(out)
```



(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

The decision rule at each split point is to split into 2 nodes based on the coordinates of (x, y). Assuming that y = 1000 is split point and for pixels(x,y), if y > 1000, pixel color = light green, if y <= 1000, pixel color = dark green.

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

The image looks the way it does because the decision tree splits pixel from (x, y) into groups by discrete thresholds. The image consists of a lot of rectangular since the patches of color are rectangular. They are arranged around the 5000 random sample.

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.

The number of patches of clor will be determined by the depth of the tree. There will be 2^d color pixels where d is the depth. The upperbound is the total number of colors of the original training set.

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.

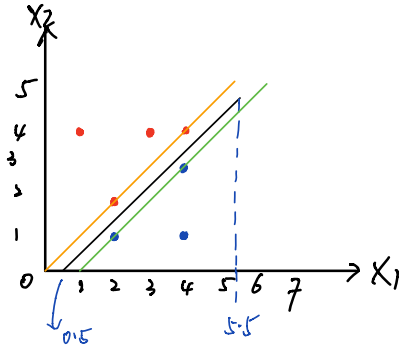
The upperbound is the number of different color pixel we have in training set. There will be $n * 2^d$ color pixels represented where n is the number of trees and d is the depth.

Written Exercises

1. Maximum-margin classifiers

obs $n = 7$ $p = 2$

a)



(0, 0.5) (5.5, 5)

$X_2 = aX_1 + b$

$\begin{cases} 0.5a + b = 0 \\ 5.5a + b = 5 \end{cases}$

$5a = 5 \quad a = 1$

$b = -0.5$

Thus, $\boxed{X_2 = X_1 - 0.5}$

$X_2 - X_1 + 0.5 = 0$

b) $\beta_0 = 0.5 \quad \beta_1 = -1 \quad \beta_2 = 1$

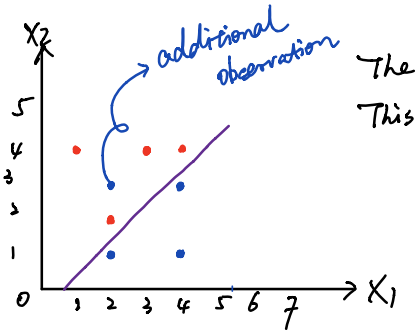
Classify to red if $0.5 - X_1 + X_2 > 0$ and classify to blue otherwise

c) Margin is the zone between the orange line and green line in the sketch above.

d) Red (2,2) Red (4,4) Blue (2,1) Blue (4,3)

e) The seventh point is Blue (4,1) which is not our support vector for the maximal margin classifier. and is far from the the hyperplane So slightly move this point will not affect our hyperplane.

f)



The new hyperplane is $X_2 = X_1 - 0.8$

This new hyperplane is closer to blue support vectors.

g) The new additional observation is Blue (7, 3)

This obs falls in the region that is classified as red

The two classes are not seperable by a hyperplane.

2. Neural networks as function approximators.

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

Recall that ReLU activation is given by

$$\sigma(x) := \begin{cases} x, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

$$f(x) = \sigma(2x-2-0) + \sigma(\tfrac{1}{3}x + \tfrac{4}{3} - 2x + 2) + \sigma(2x-7-\tfrac{1}{3}x-\tfrac{4}{3}) + \sigma(-\tfrac{5}{3}x+15-2x+7) + \sigma(\tfrac{5}{3}x-15)$$

$$= \sigma(2x-2) + \sigma(-\tfrac{5}{3}x+\tfrac{10}{3}) + \sigma(\tfrac{5}{3}x-\tfrac{25}{3}) + \sigma(-\tfrac{11}{3}x+22) + \sigma(\tfrac{5}{3}x-15)$$

$$= 2\sigma(x-1) - \tfrac{5}{3}\sigma(x-2) + \tfrac{5}{3}\sigma(x-5) - \tfrac{11}{3}\sigma(x-6) + \tfrac{5}{3}\sigma(x-9)$$

$$W_1 = [1, 1, 1, 1, 1]$$

$$b_1 = [-1, -2, -5, -6, -9]$$

$$w_2 = [2, -\frac{5}{3}, \frac{5}{8}, -\frac{11}{3}, \frac{5}{3}]$$

Thus, there should be five neurons in 1 hidden layer