PRAFULLA KUMAR CHOUBEY – 32500790

Question-1 **Question : 1**

$P(\neg J, \neg M, \neg A, \neg E, \neg B)$

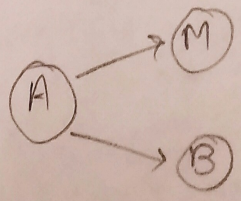
$= P(\neg B) \cdot P(\neg E) \cdot P(\neg A | \neg B, \neg E) \cdot P(\neg J | \neg A) \cdot P(\neg M | \neg A)$

$= 0.999 \times 0.998 \times 0.999 \times 0.95 \times 0.99$

$= 0.936743$

Question-2 **Question : 2**

(i)



Alarm → A

Mary Calls → M

Burglary → B

(ii) (a) Node 1 → (A)

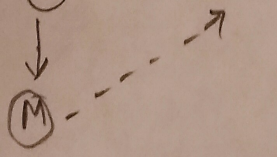
(b) Node 2 → (A) - - -> (M) , parent(M) = A

∵ Mary agreed to call if she hears the Alarm, Mary calls or not depends on whether alarm happened or not.

∴ $P(M|A) \neq P(M)$

(A) ⟶ (M)

(c) Node 3 →



Parent(B) = A, M

$$P(B|A,M) = \frac{P(M|A,B) \cdot P(B|A)}{P(M|A)}$$

Given Alarm, Marycalls is independent of burglary. So, $P(M|AB) = P(M|A)$

$\Rightarrow P(B|AM) = P(B|A)$ ∴ (B) ← (A)

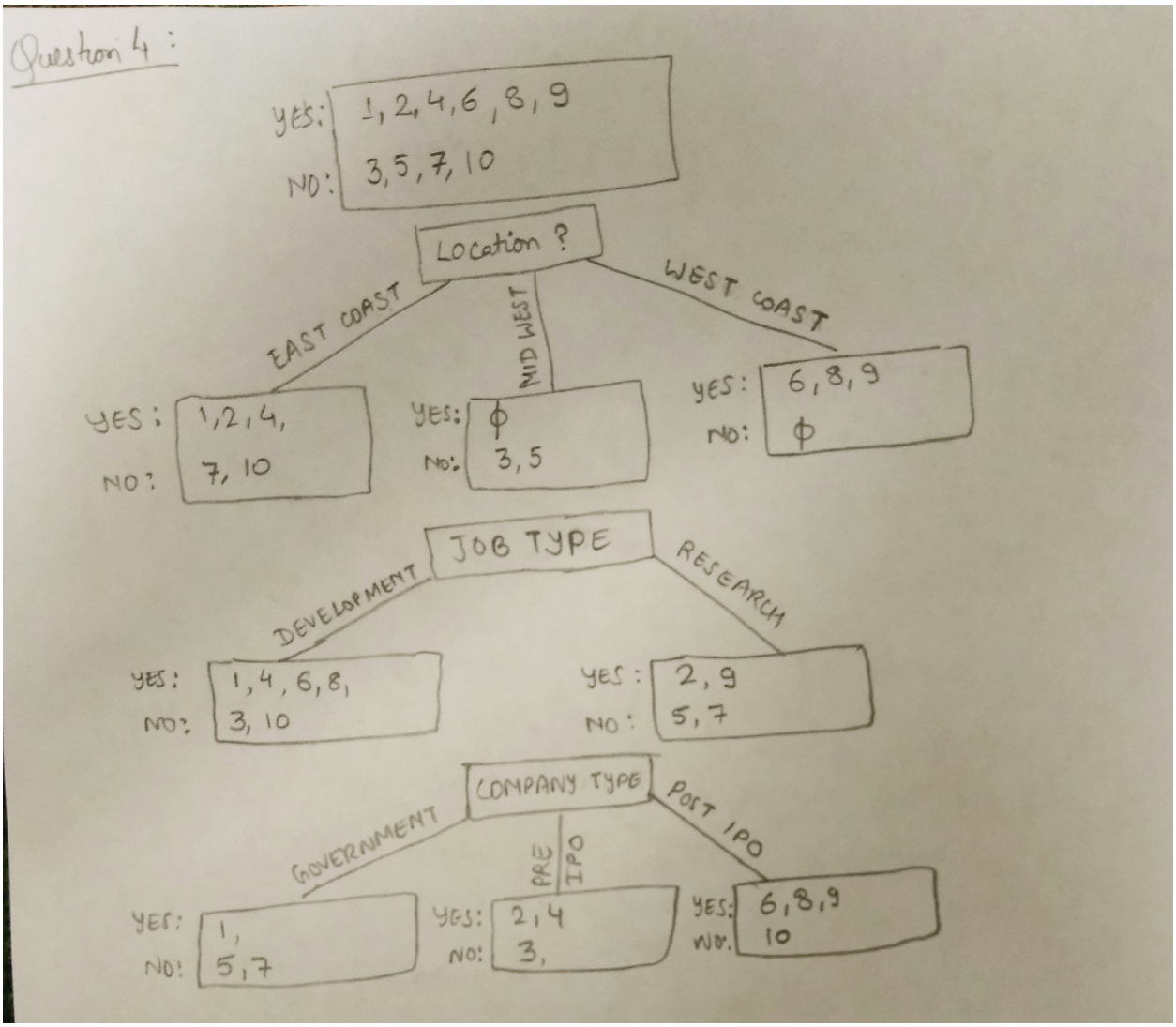
Question 3:

P(Burglary|MaryCalls) > P(Burglary|MaryCalls, Earthquake)

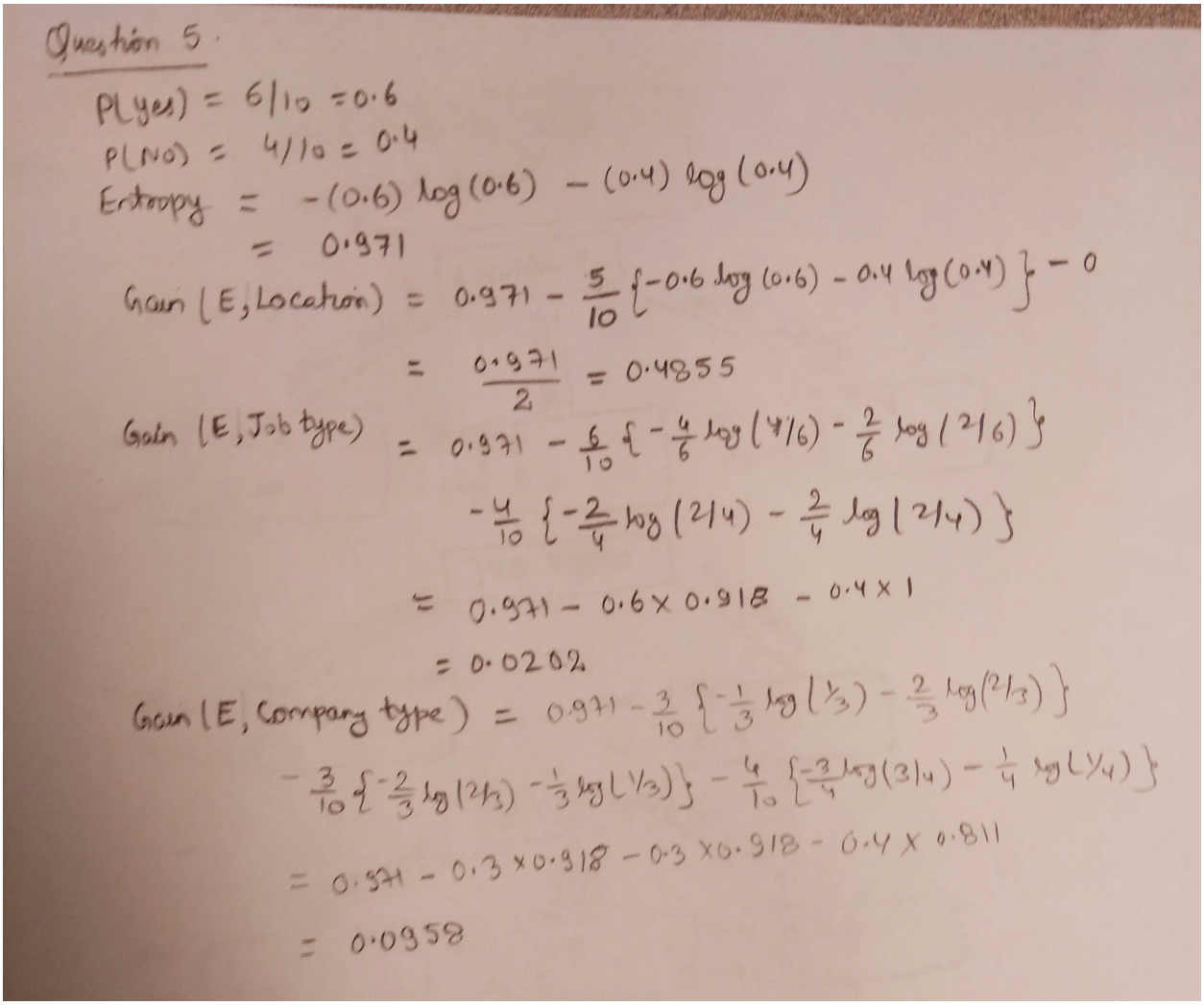
Here, Burglary and Earthquake are causes and Marycalls is the effect. Although Earthquake and Burglary are independent of each other, the presence of one cause (Earthquake) makes the other (Burglary) unlikely. This phenomenon is called “explaining away” or Inter-causal inference.

MaryCalls event depends on whether Alarm happened or not which in turn, depends on Burglary or Earthquake. So when it is known that Earthquake (cause II) and Marycalls (Effect) have happened, it is less likely that Burglary (cause I) has happened. However, when there is no detail on the occurrence of Earthquake, likelihood of Burglary given Marycalls is greater.

Question 4:



Question 5:



Question 6:

'Location' attribute should be chosen first.

Information Gain(E, attribute) = Entropy(E) – Entropy(E| attribute)

Information gain value is the mutual information of 'E' and 'attribute', i.e. the reduction in the entropy of 'E' achieved by learning the state of the random variable 'attribute'.

At each depth, we choose the attribute which most rapidly narrow down the state of 'E'. And so, attribute with highest mutual information is chosen first. In the above case, 'Location' has the highest entropy reduction (information gain) and so it will be chosen over 'Job Type' and 'Company Type'.

Question 7:

Code:

```
from __future__ import division
import math

def calculate_entropy(sample):
    total = sample[0] + sample[1]
    result = 0
    for s in sample:
        val = s/total
        if val == 0: continue
        result += - (val)*math.log(val,2)

    return result

def information_gain(sample, sampleRes):
    total_entropy = calculate_entropy(sample)
    total = sample[0] + sample[1]

    result = 0
    gain = total_entropy

    for s in sampleRes:
        val = (s[0]+s[1])/total
        gain -= val*calculate_entropy(s)

    return gain

print('{:20s}'.format("Attribute") + "Information Gain")
print("================================")
print('{:20s}'.format("Location") + str(information_gain([6,4],[[3,2],[0,2],[3,0]])))
print('{:20s}'.format("Job Type") + str(information_gain([6,4],[[4,2],[2,2]])))
print('{:20s}'.format("Company Type") + str(information_gain([6,4],[[1,2],[5,2]])))
print
print ("Slide 07, Page 09")
print('{:20s}'.format("Attribute") + "Information Gain")
print("================================")
print('{:20s}'.format("Alternate") + str(information_gain([6,6],[[3,3],[3,3]])))
print('{:20s}'.format("Bar") + str(information_gain([6,6],[[3,3],[3,3]])))
print('{:20s}'.format("Fri/Sat") + str(information_gain([6,6],[[2,3],[4,3]])))
print('{:20s}'.format("Hungry") + str(information_gain([6,6],[[5,2],[1,4]])))
print('{:20s}'.format("Patrons") + str(information_gain([6,6],[[0,2],[4,0],[2,4]])))
print('{:20s}'.format("Price") + str(information_gain([6,6],[[3,4],[2,0],[1,2]])))
print('{:20s}'.format("Rain") + str(information_gain([6,6],[[2,2],[4,4]])))
print('{:20s}'.format("Reservation") + str(information_gain([6,6],[[3,2],[3,4]])))
print('{:20s}'.format("Type") +
str(information_gain([6,6],[[1,1],[1,1],[2,2],[2,2]])))
print('{:20s}'.format("Wait Estimate") +
str(information_gain([6,6],[[4,2],[1,1],[1,1],[0,2]])))
```

Output:

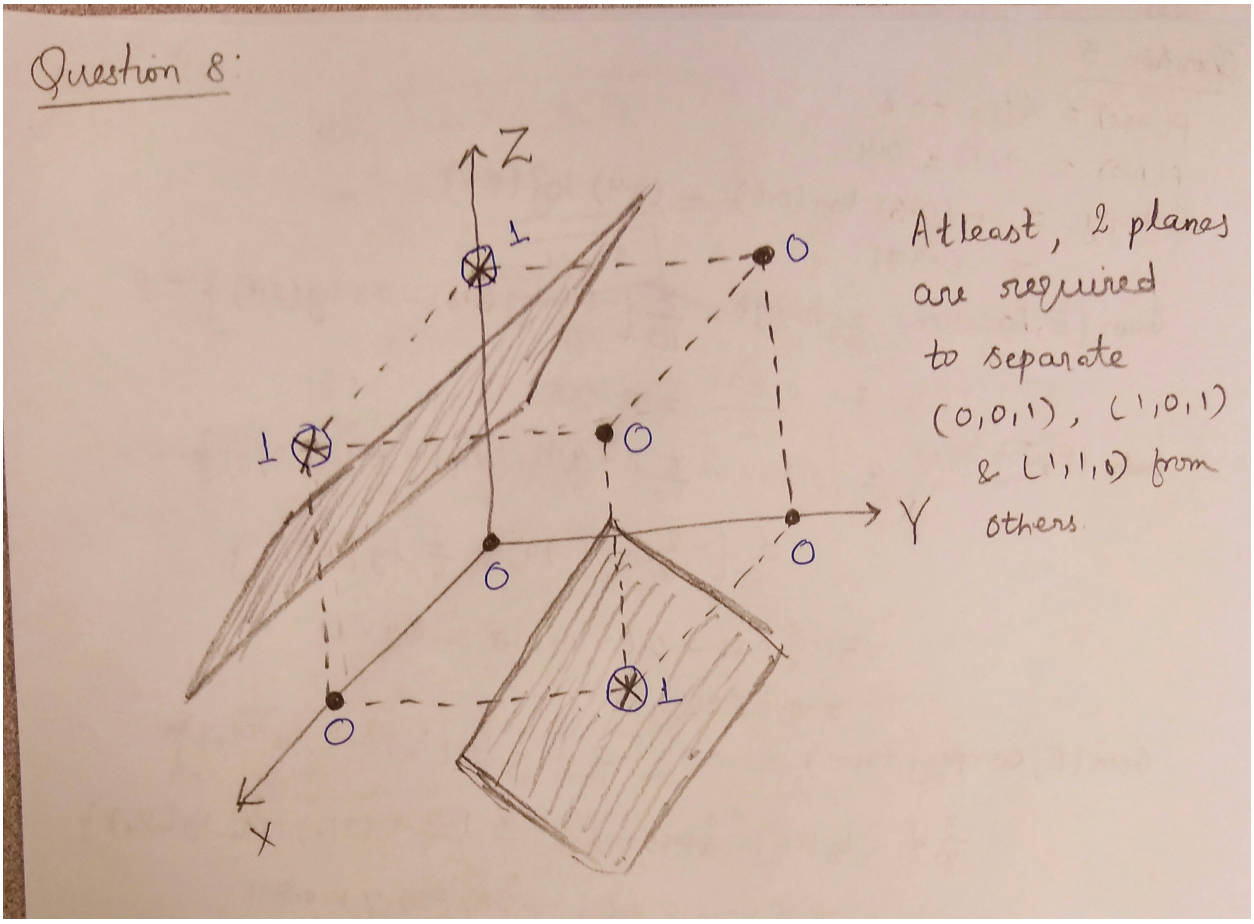
```
Attribute           Information Gain
================================
Location            0.485475297227
Job Type            0.019973094022
Company Type        0.0912774462417
```

Slide 07, Page 09

Attribute	Information Gain
Alternate	0.0
Bar	0.0
Fri/Sat	0.0207208396239
Hungry	0.1957096288
Patrons	0.540852082973
Price	0.1957096288
Rain	1.11022302463e-16
Reservation	0.0207208396239
Type	1.11022302463e-16
Wait Estimate	0.207518749639

Question 8:

(1) No, a single perceptron unit cannot solve the following Classification problem.
(2) As illustrated in the figure below, no single plane can be used to separate points {(0,0,1), (1,0,1), (1,1,0)} from points {(0,0,0), (0,1,0), (0,1,1), (1,0,0), (1,1,1)}.



(3) As shown in figure above, no single plane exists which can separate points {(0,0,1), (1,0,1), (1,1,0)} from points {(0,0,0), (0,1,0), (0,1,1), (1,0,0), (1,1,1)}. We need atleast two planes to perform this classification.
The best single plane that can be found in this case will have 1 False Negative (top plane) i.e. point (1,1,0) will be misclassified.

Question 9:

Code:

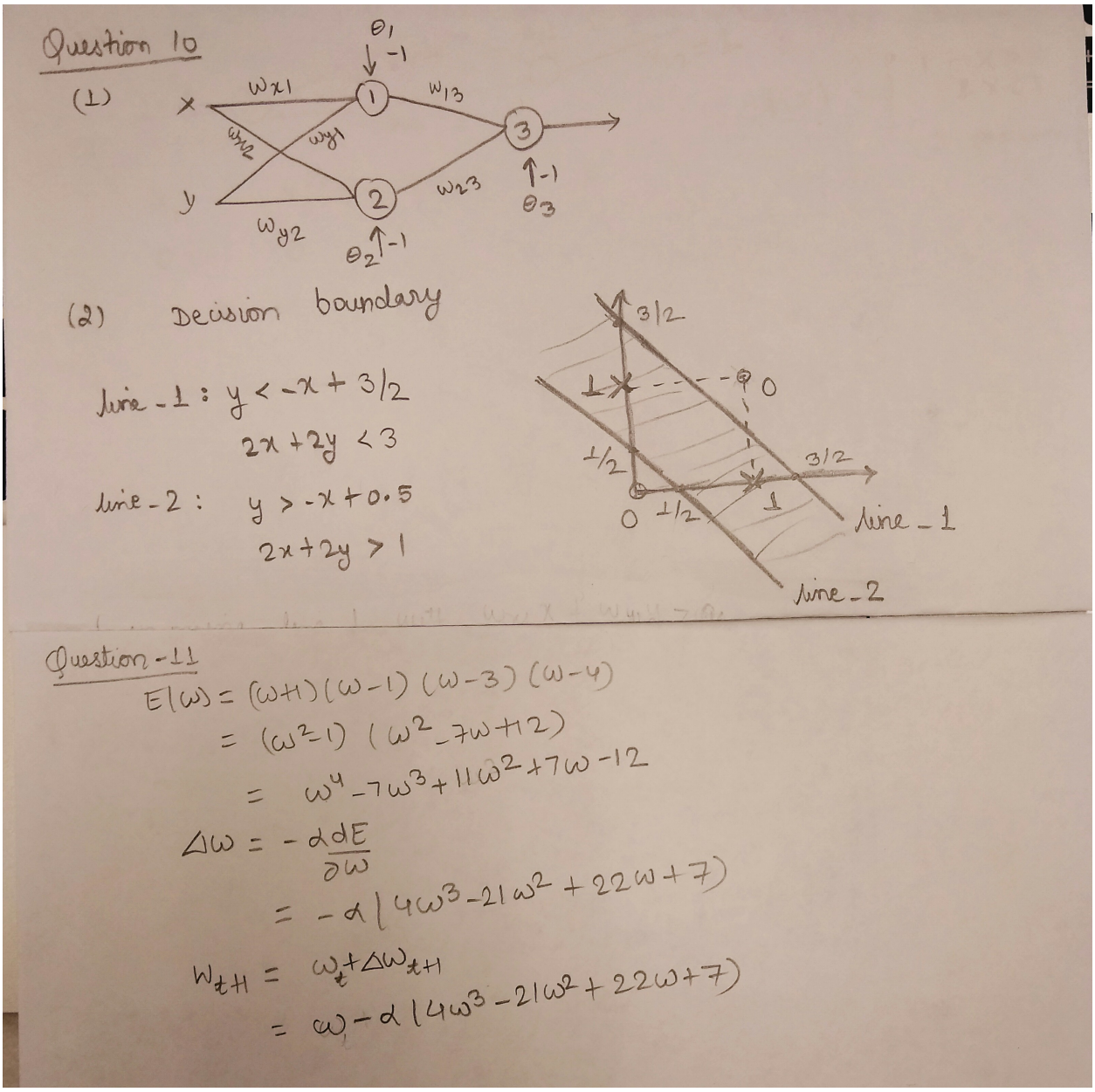
```
import numpy as np
import random
import os, subprocess
import matplotlib.pyplot as plt
from matplotlib import animation

images = []

class Perceptron:
    def __init__(self, N):
    # Random linearly separated data
        xA,yA,xB,yB = [random.uniform(-1, 1) for i in range(4)]
        self.V = np.array([xB*yA-xA*yB, yB-yA, xA-xB])
```

```
    self.X = self.generate_points(N)

def trainingDataAnd(self):
    t = 1.5
    self.V = np.array([t,1,1])
    X = []
    N = len(self.X)
    for i in range(N):
        x1, x2 = [random.uniform(-1, 1) for i in range(2)]
        x = np.array([1, x1, x2])
        if self.V.T.dot(x) >= 0:
            s = 1
        else:
            s = -1
        X.append((x, s))
    self.X = X

def trainingDataOr(self):
    t = 0.5
    self.V = np.array([t,1,1])
    X = []
    N = len(self.X)
    for i in range(N):
        x1, x2 = [random.uniform(-1, 1) for i in range(2)]
        x = np.array([1, x1, x2])
        if self.V.T.dot(x) >= 0:
            s = 1
        else:
            s = -1
        X.append((x, s))
    self.X = X

def trainingDataNot(self):
    t = -0.5
    self.V = np.array([t,-1,0])
    X = []
    N = len(self.X)
    for i in range(N):
        x1, x2 = [random.uniform(-1, 1) for i in range(2)]
        x = np.array([1, x1, x2])
        if self.V.T.dot(x) >= 0:
            s = 1
        else:
            s = -1
        X.append((x, s))
    self.X = X

def DataXor(self):
    t = 0
    # self.V = np.array([t,-1,0])
    X = []
    N = len(self.X)
    for i in range(N):
        x1, x2 = [random.uniform(-1, 1) for i in range(2)]
        x = np.array([1, x1, x2])
        if (x1 >= 0 and x2 >= 0) or (x1 < 0 and x2 < 0):
            s = -1
        else:
            s = 1
        X.append((x, s))
    self.X = X
```

```
def generate_points(self, N):
    X = []
    for i in range(N):
        x1,x2 = [random.uniform(-1, 1) for i in range(2)]
        x = np.array([1,x1,x2])
        s = int(np.sign(self.V.T.dot(x)))
        X.append((x, s))
    return X

def plot(self, mispts=None, vec=None, save=False):
    fig = plt.figure(figsize=(5,5))
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    V = self.V
    a, b = -V[1]/V[2], -V[0]/V[2]
    l = np.linspace(-1,1)
    plt.plot(l, a*l+b, 'k-')
    cols = {1: 'r', -1: 'b'}
    for x,s in self.X:
        plt.plot(x[1], x[2], cols[s]+'o')
    if mispts:
        for x,s in mispts:
            plt.plot(x[1], x[2], cols[s]+'.')
    if vec != None:
        aa, bb = -vec[1]/vec[2], -vec[0]/vec[2]
        plt.plot(l, aa*l+bb, 'g-', lw=2)
    if save:
        if not mispts:
            plt.title('N = %s' % (str(len(self.X))))
        else:
            plt.title('N = %s with %s test points' \
            % (str(len(self.X)),str(len(mispts))))
        plt.savefig('p_N%s' % (str(len(self.X))), \
        dpi=200, bbox_inches='tight')
    # plt.show()

def classification_error(self, vec, pts=None):
# Error defined as fraction of misclassified points
    if not pts:
        pts = self.X
    M = len(pts)
    n_mispts = 0
    for x,s in pts:
        if int(np.sign(vec.T.dot(x))) != s:
            n_mispts += 1
    error = n_mispts / float(M)
    return error

def choose_miscl_point(self, vec):
# Choose a random point among the misclassified
    pts = self.X
    mispts = []
    for x,s in pts:
        if int(np.sign(vec.T.dot(x))) != s:
            mispts.append((x, s))
    return mispts[random.randrange(0,len(mispts))]

def pla(self, save=False):
# Initialize the weigths to zeros
    w = np.zeros(3)
```

```
        X, N = self.X, len(self.X)
        it = 0
        # Iterate until all points are correctly classified
        while self.classification_error(w) != 0 and it <= 100:
            it += 1
            # Pick random misclassified point
            x, s = self.choose_miscl_point(w)
            # Update weights
            w += s*x
            if save:
                self.plot(vec=w)
                plt.title('N = %s, Iteration %s\n' \
                % (str(N),str(it)))
                plt.savefig('p_N%s_it%s' % (str(N),str(it)), \
                dpi=200, bbox_inches='tight')
                # images.append(plt)
        self.w = w

    def check_error(self, M, vec):
        check_pts = self.generate_points(M)
        return self.classification_error(vec, pts=check_pts)

def plotXOR(pAnd, pNot, pOr, mispts=None, vec=None, save=False,):
    fig = plt.figure(figsize=(5, 5))
    plt.xlim(-1, 1)
    plt.ylim(-1, 1)
    if mispts:
        for x,s in mispts:
            val1 = np.sign(pNot.w[1]*(pAnd.w[1] * x[1] + pAnd.w[2] * x[2]))
            val2 = np.sign(pOr.w[1] * x[1] + pOr.w[2]*x[2])
            val3 = np.sign(pAnd.w[1]*val1 + pAnd.w[2]*val2)
            if val3 == s:
                plt.plot(x[1], x[2], 'ob')
            else:
                plt.plot(x[1], x[2], 'or')
    plt.title('N = %s, Iteration %s\n' \
              % (str(10), str(5)))
    plt.savefig('p_N%s_it%s' % (str(10), str(5)), \
                dpi=200, bbox_inches='tight')
    plt.show()

p = Perceptron(100)
p.trainingDataAnd()
p.pla(save=True)
p1 = Perceptron(100)
p1.trainingDataNot()
p1.pla(save=False)
p2 = Perceptron(100)
p2.trainingDataOr()
p2.pla(save=False)
pf = Perceptron(500)
pf.DataXor()
plotXOR(p,p1,p2,pf.X)
pf.pla(save=True)
p.plot(p.generate_points(500), p.w, save=True)
err = []
for i in range(100):
    err.append(p.check_error(500, p.w))
print(np.mean(err))
```

Question 10:



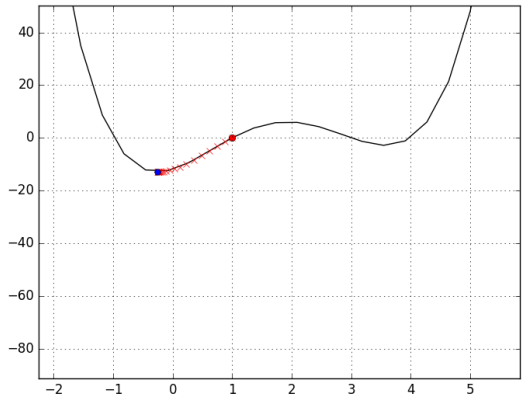
Question 11: In image above

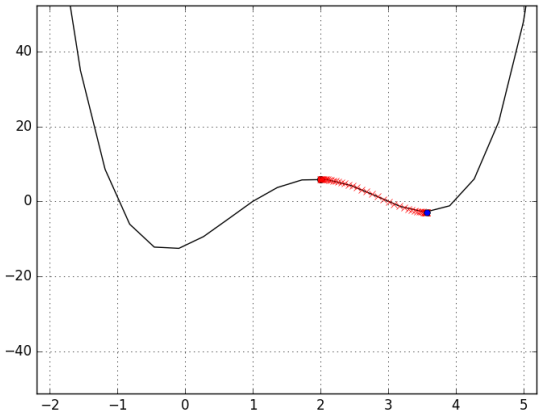
Question 12:

Code:

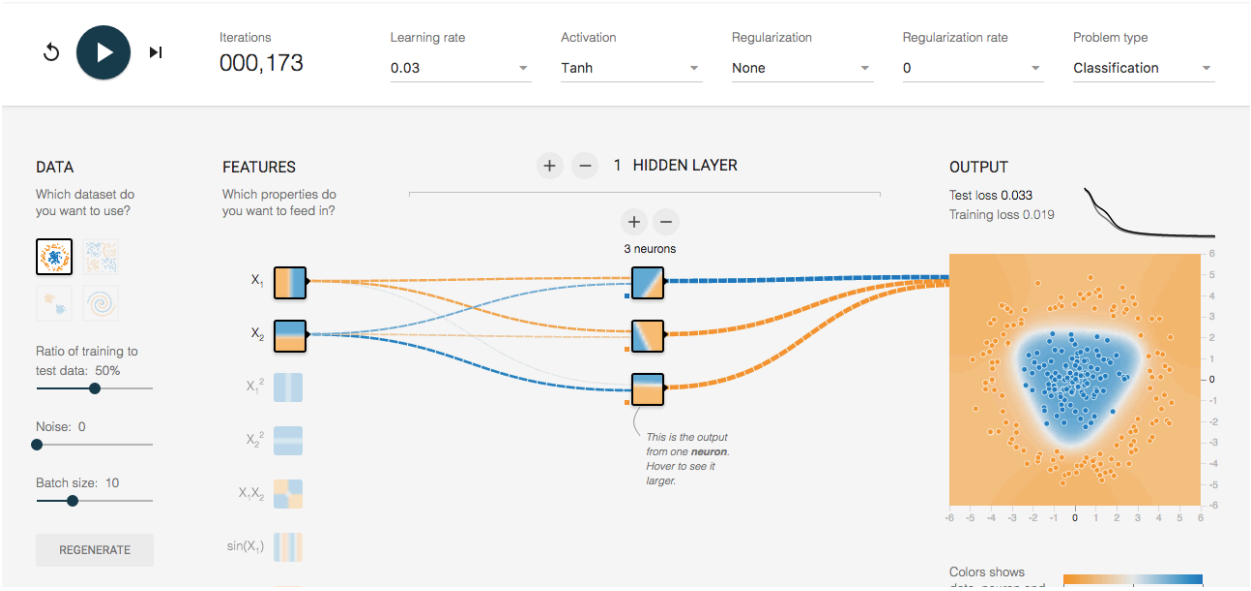
```
import matplotlib.pyplot as plt
w_old = 0
w = -2
step_size = 0.01
precision = 0.001

def f_derivative(x):
    return (4 * (x ** 3)) - (21 * (x ** 2)) + (22 * x) + 7

def fx(x):
    return (x+1)*(x-1)*(x-3)*(x-4)

xarr=[]
yarr=[]
while abs(w - w_old) > precision:
    w_old = w
    gradient = f_derivative(w_old)
    w_delta = gradient * step_size
    w = w_old - w_delta
    xarr.append(w_old)
    yarr.append(fx(w))
print xarr[:10]
plt.plot(xarr,yarr)
plt.ylabel('Error')
plt.xlabel('(w)')
plt.show()

print("Local minimum occurs at {}.".format(round(w, 2)))
```

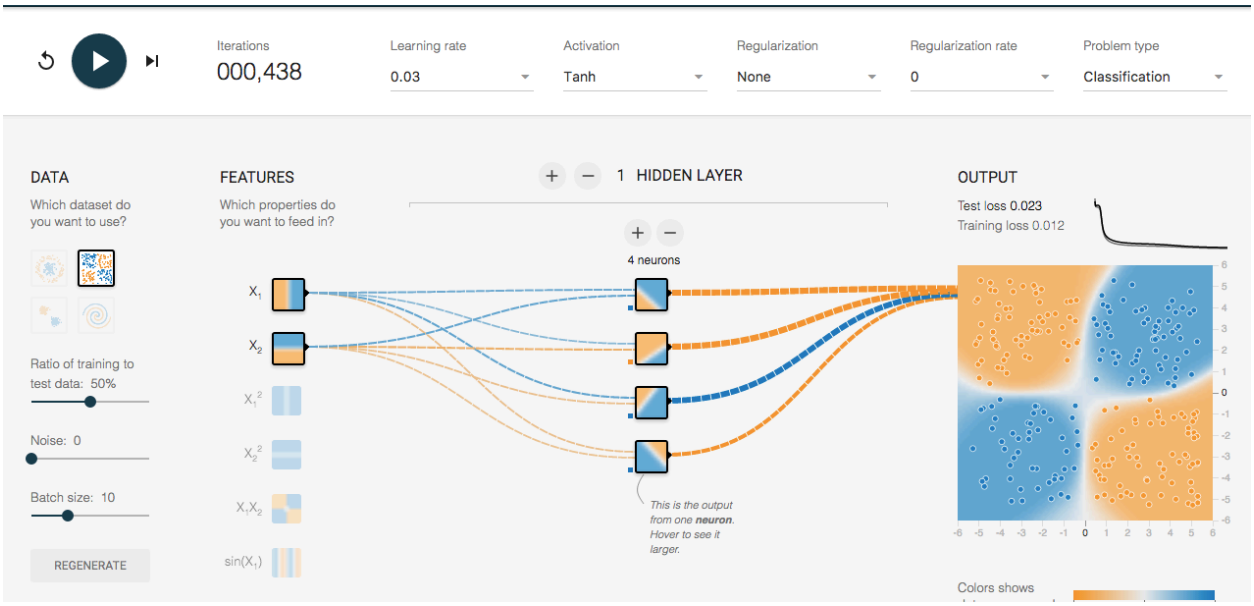
Plots:

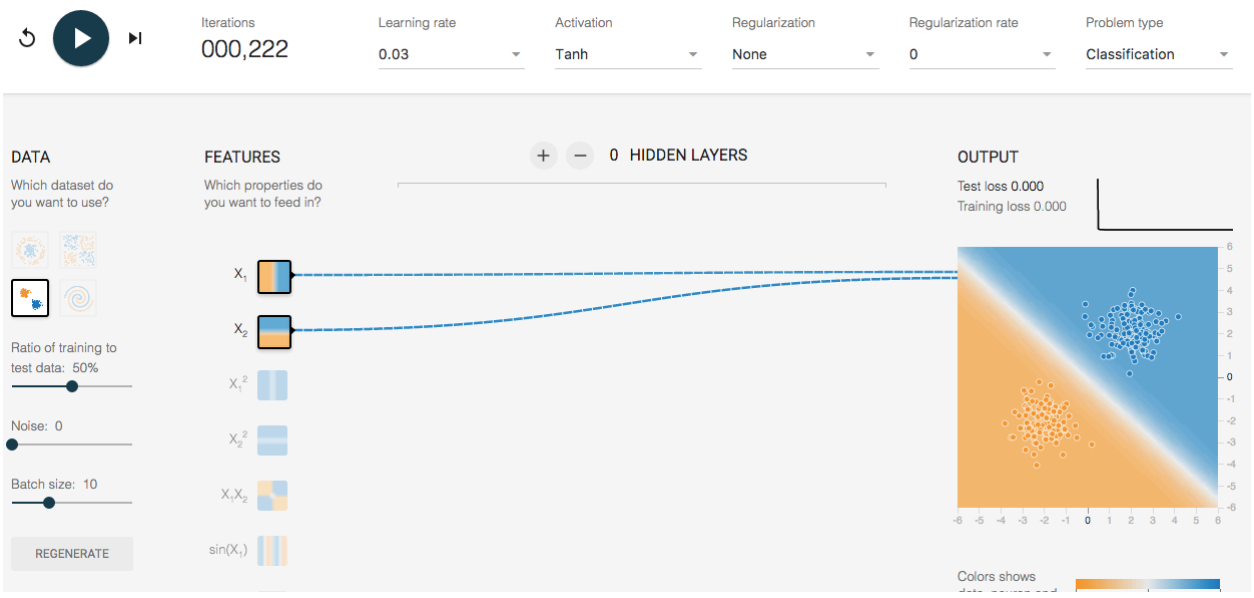


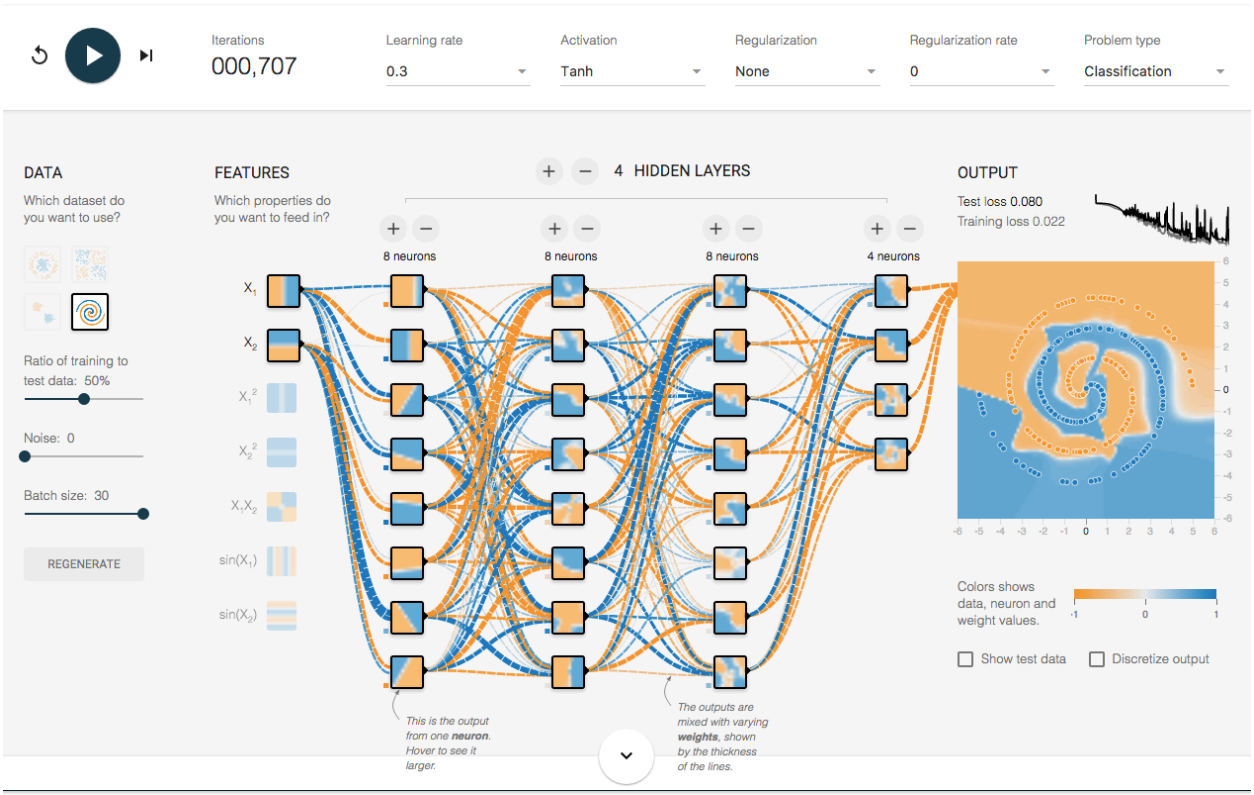

40
20
0
−20
−40
−2
−1
0
1
2
3
4
5

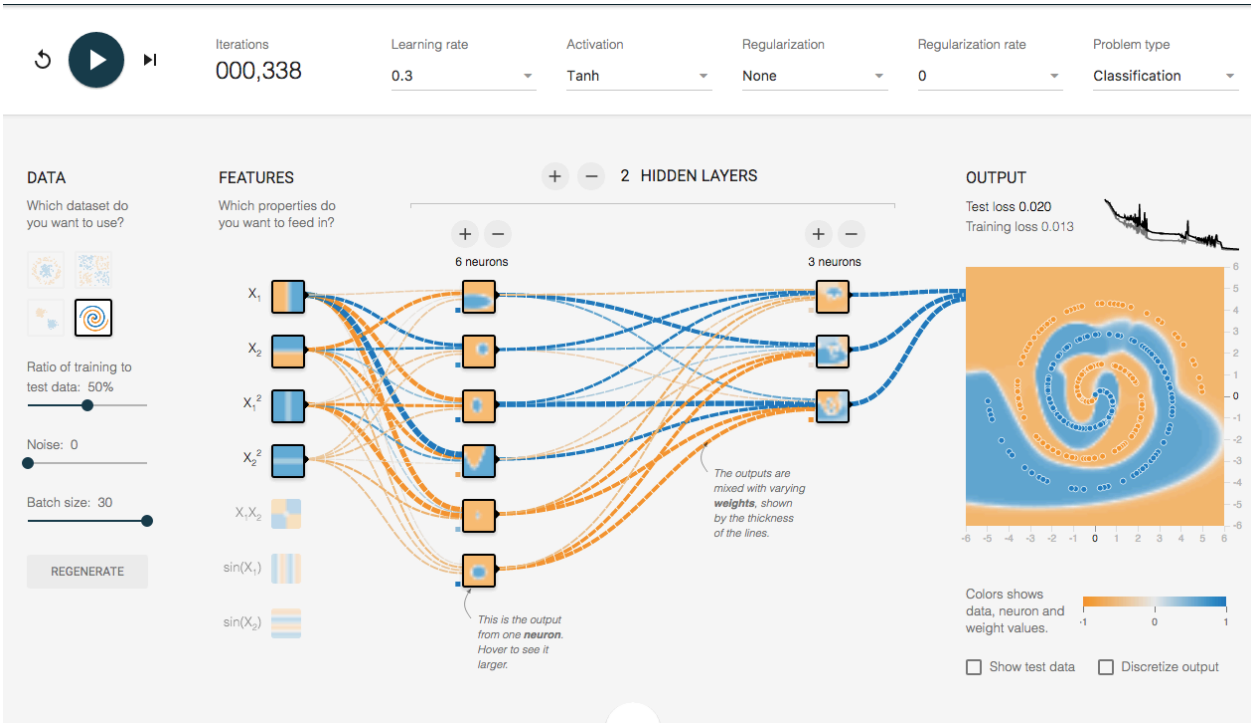
Question 13:


Iterations
000,173
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1²
X2²
X1X2
sin(X1)
1 HIDDEN LAYER
3 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.033
Training loss 0.019
Colors shows


Iterations
000,438
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
4 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.023
Training loss 0.012
Colors shows


Iterations
000,222
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
0 HIDDEN LAYERS
OUTPUT
Test loss 0.000
Training loss 0.000
Colors shows


Iterations
000,707
Learning rate
0.3
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 30
REGENERATE
FEATURES
Which properties do you want to feed in?
4 HIDDEN LAYERS
8 neurons
8 neurons
8 neurons
4 neurons
X1
X2
X1²
X2²
X1X2
sin(X1)
sin(X2)
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.080
Training loss 0.022
Colors shows data, neuron and weight values.
Show test data
Discretize output


Iterations
000,338
Learning rate
0.3
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 30
REGENERATE
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
6 neurons
3 neurons
X1
X2
X1²
X2²
X1X2
sin(X1)
sin(X2)
The outputs are mixed with varying weights, shown by the thickness of the lines.
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.020
Training loss 0.013
Colors shows data, neuron and weight values.
Show test data
Discretize output

Question 15:

Question 15: $\bar{w}_j \leftarrow \bar{w}_j + \eta h(j, i(\bar{x}))(\bar{x} - w_j)$

(i) $\eta = 1$

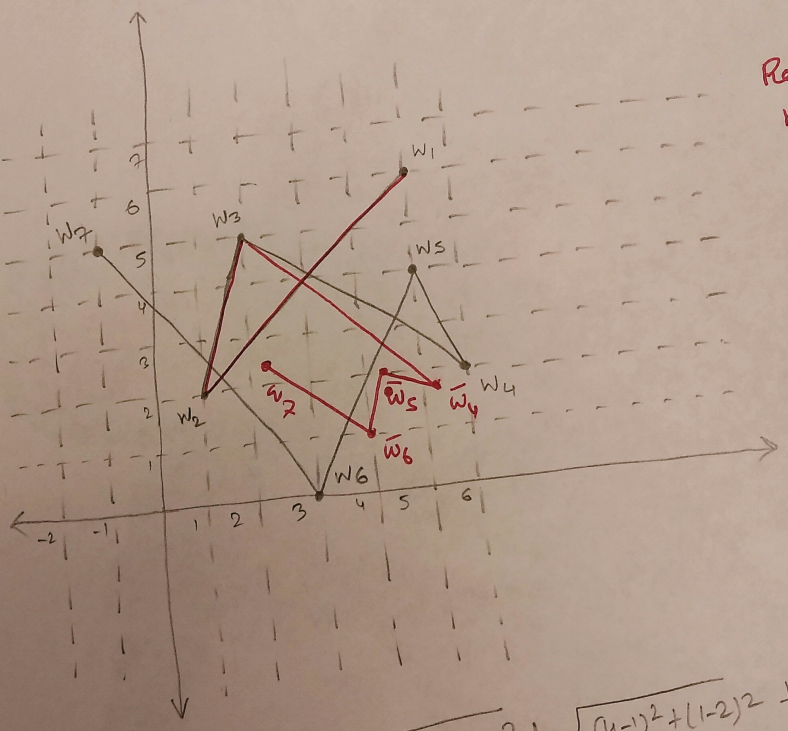
(ii) $h(j, i(\bar{x})) = 1$ if $j = i(\bar{x})$

$= 2/3$ if $j = i(\bar{x}) \pm 1$

$= 1/3$ if $j = i(\bar{x}) \pm 2$

(iii)

	$\bar{w}_1$	$\bar{w}_2$	$\bar{w}_3$	$\bar{w}_4$	$\bar{w}_5$	$\bar{w}_6$	$\bar{w}_7$
			2	6	5	3	-1
w_{i1}	5	1	5	2	4	0	5
w_{i2}	6	2					



Red connections are new one.

(iv) $\bar{x} = (4, 1)$

$BMU = \text{argmin}\{\sqrt{(5-4)^2+(6-1)^2}, \sqrt{(4-1)^2+(1-2)^2}, \sqrt{(4-2)^2+(5-1)^2}, \sqrt{(6-4)^2+(2-1)^2}, \sqrt{(5-4)^2+(4-1)^2}, \sqrt{(4-3)^2+(1-0)^2}, \sqrt{(4+1)^2+(5-1)^2}\}$

$BMU = \bar{w}_6$

Iteration : 1 →

$\bar{w}_6 \leftarrow (3,0) + 1 \times 1 \times (4-3, 1-0)$

$\Rightarrow \bar{w}_6 = (4, 1)$

$\bar{w}_5 \leftarrow (5,4) + 1 \times \frac{2}{3} \times (4-5, 1-4)$

$\Rightarrow \bar{w}_5 = \left(\frac{13}{3}, 2\right) = (4.33, 2)$

$\bar{w}_7 \leftarrow (-1,5) + 1 \times \frac{2}{3} \times (4+1, 1-5)$

$\Rightarrow \bar{w}_7 = \left(\frac{7}{3}, \frac{7}{3}\right) = (2.33, 2.33)$

$\bar{w}_4 \leftarrow (6,2) + 1 \times \frac{1}{3} \times (4-6, 1-2)$

$\Rightarrow \bar{w}_4 = \left(\frac{16}{3}, \frac{5}{3}\right) = (5.33, 1.66)$

$\bar{w}_1$, $\bar{w}_2$ and $\bar{w}_3$ will not change as $h(j, i(x)) = 0$

if $\{j > i(x) \pm 2\}$

New weights.

	w_1	w_2	w_3	w_4	w_5	w_6	w_7
w_{i1}	5	1	2	5.33	4.33	4	2.33
w_{i2}	6	2	5	1.66	2	1	2.33