Asam Mahmood

Question 1 [4 pts]: Figure 1 shows some training samples in two-dimensional space and have three classes: crosses, circles, and diamonds. Designing a feedforward neural network to separate samples as showing in Figure 1.

- Show network structure in terms of input, output, and hidden layers (if any). (Simplify each neuron as a circle. No need to show details inside each neuron) [2 pts].
- Explain the role of each layer (including input layer, hidden layer (if any), and output layer) [1 pt],
- Draw decision surface of the network [1 pt].

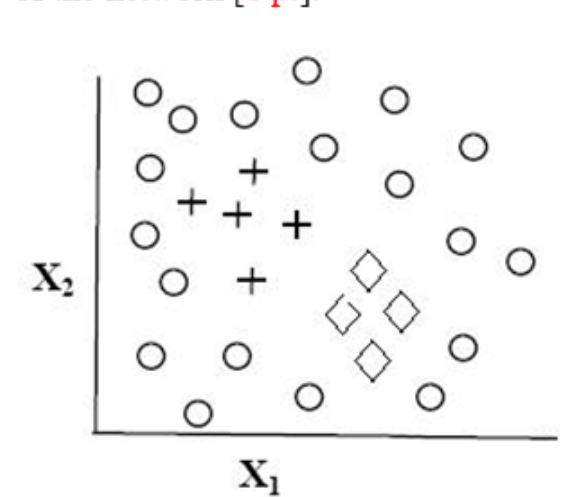


Figure 1

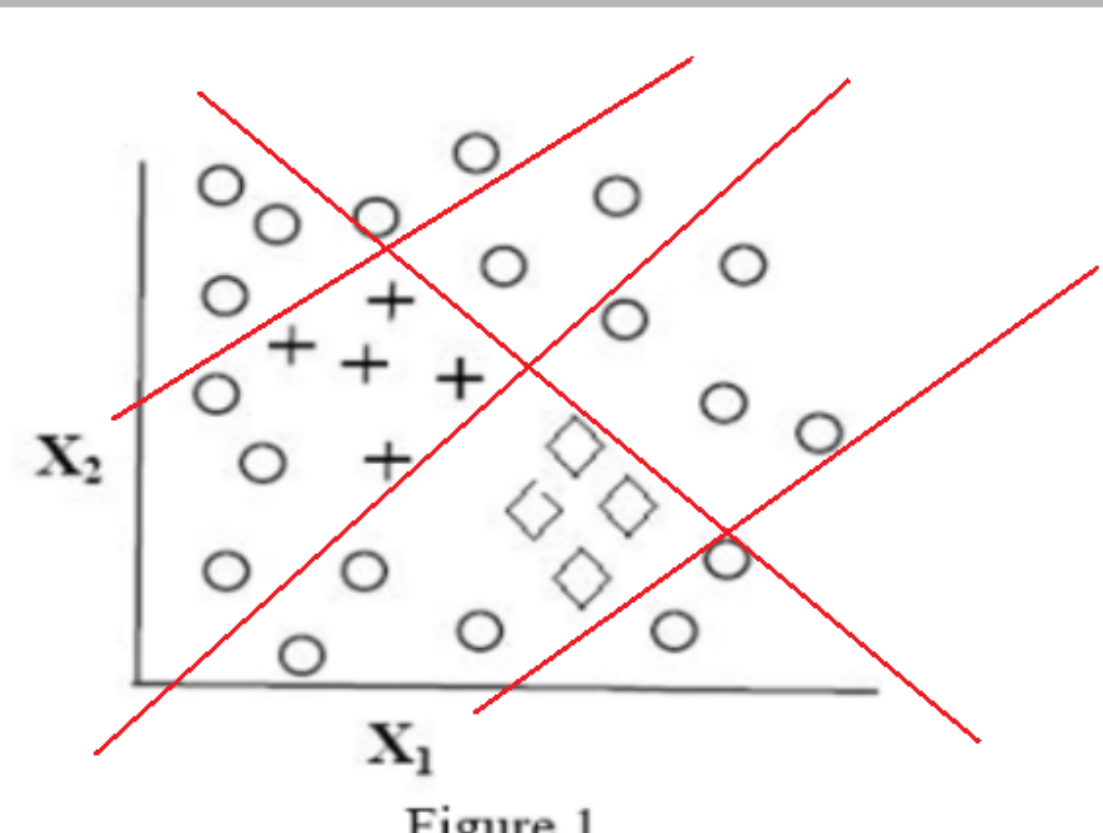
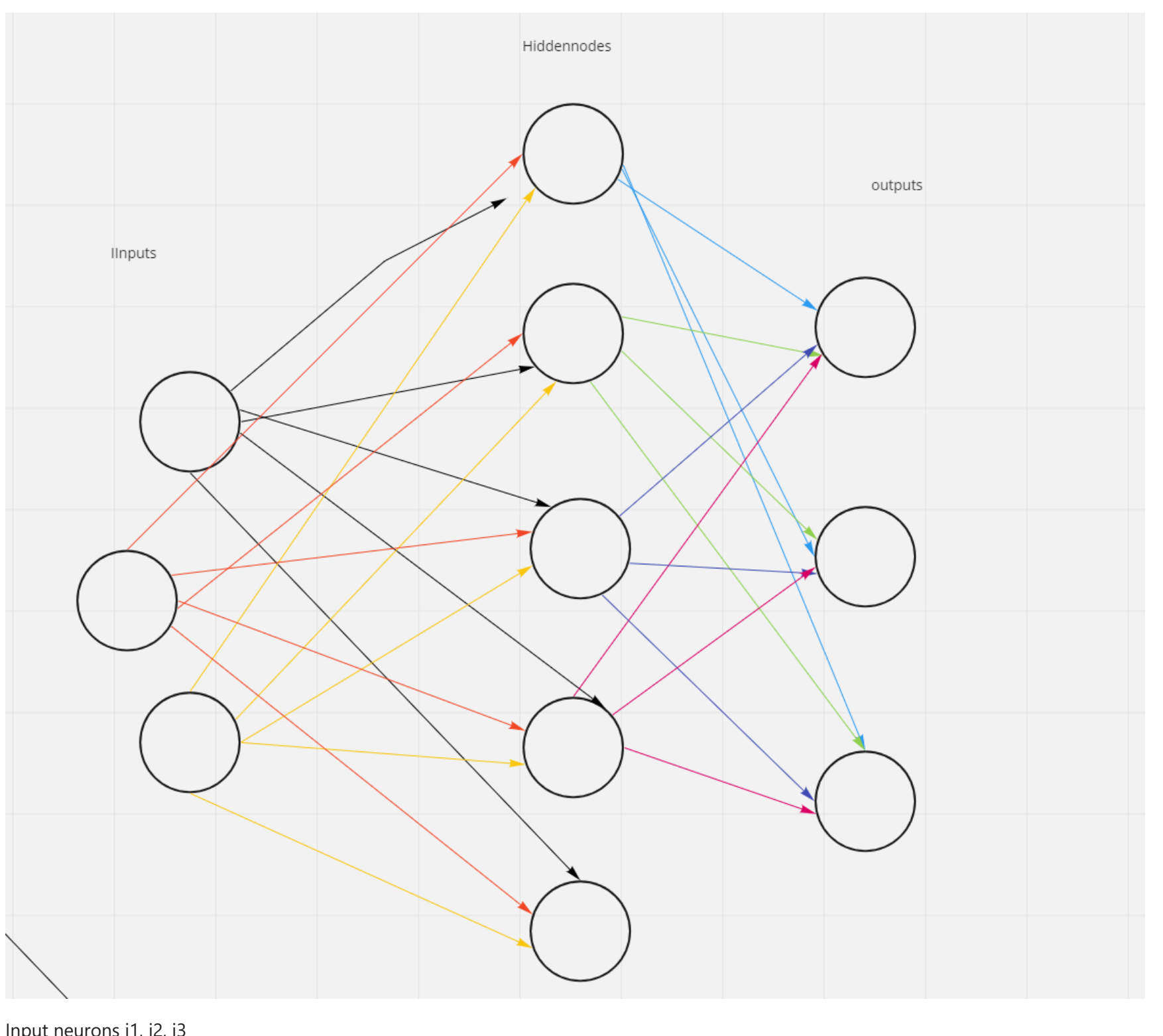


Figure 1



Input neurons i1, i2, i3

Input weights w1, w2, w3,w4,w5

Output neuron Oi1, oi2, oi3

Working The basic unit of computation in a neural network is the neuron, often called a node or unit. It receives input from some other nodes, or from an external source and computes an output. Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs.

1. Input Nodes – The Input nodes provide information from the outside world to the network and are together referred to as the "Input Layer". No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes.
2. Hidden Nodes – The Hidden nodes have no direct connection with the outside world (hence the name "hidden"). They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a "Hidden Layer". While a feedforward network will only have a single input layer and a single output layer, it can have zero or multiple Hidden Layers.
3. Output Nodes – The Output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network to the outside world.
4. Bias Nodes - The bias node in a neural network is a node that is always 'on'. That is, its value is set to 1 without regard for the data in a given pattern. It is analogous to the intercept in a regression model, and serves the same function.

Question 2 [4 pts]: Figure 2 shows a single layer neural network with m+1 weight values (including bias). Assume a training instance [x(n), d(n)] is used for training, where x(n)=<$x_1(n)$, $x_2(n)$,…, $x_i(n)$,…, $x_m(n)$> are the feature values, and d(n) is the lable of the instance. Use gradient descent learning to derive weight updating rule for w_i.

- Using instance x(n) as input, show local field (v) and output of the neuron, respectively [1 pt]
- Using instance x(n) as input, show squared error of the network [1 pt]
- Show derivations and final weight updating rule for weight w_i [2 pts]

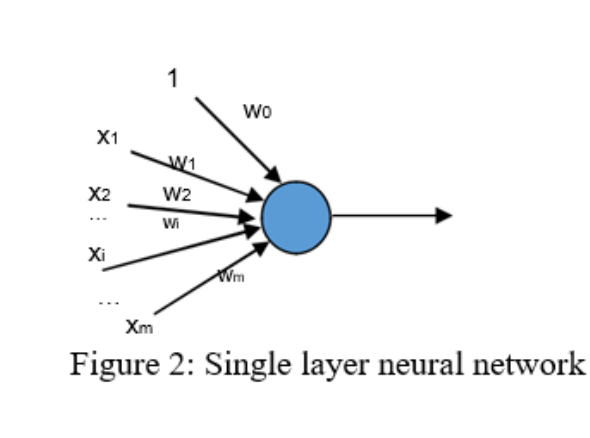


Figure 2: Single layer neural network

Q2 A

$V = w^T x$ $\quad \vec{V} = (w_0 + w_1 + \dots + w_{m+1})$

$\vec{X} = (X_1 + X_2 + \dots X_m)$

$\vec{X}_{m+1} = (1, X_1, X_2 \dots X_{m+1})$

$V = w_0 + w_1 X_1 + w_2 X_2 \dots W_{m+1} X_{m+1}$

local field

Q2 B

$E_i = d_i - O_i$ $\qquad O_i = f(v_i)$ ↑ Activation

$E_i^2 = (d_i - O_i)^2$

$= \frac{\sum (d_i - o_i)^2}{n} = \frac{1}{n}\sum (d_i - o_i)^2$

Q2C $\frac{\partial F(w)}{\partial w_i} = X(n)$

$i_{N} \neq 0$

$i \neq 0$

$\frac{\partial F(w_0, w_1, w_2)}{\partial w_1} = \frac{\frac{1}{2}[d(n) - w_0 - w_1 x_1(n) - w_2 x_2(n)]^2}{\partial w_1}$

$= [d(n) - w_0 - w_1 x_1(n) - w_2 x_2(n)] \times (-1)$

$= [d(n) - w_0 - w_1 x_1(n) - w_2 x_2(n)] \cdot x_i(n)$

Question 3 [4 pts]: Given the following six instances each with five attributes (Outlook, Temperature, Humidity, Wind, Day) and one class label,

- calculate **Entropy** of the whole system [0.5 pt]
- calculate **Information gain** for attribute "Outlook" [0.5 pt]
- calculate **Gini-index** for attribute "Outlook" [1 pt]
- Calculate information gain and Gini-index for attribute "Day" [1 pt]
- Is "Day" a good feature being used as the root node of a decision tree. Why or why not? How to avoid using "Day" as the root node to create the tree [1 pt]

ID	Outlook	Temperature	Humidity	Wind	Day	Class
1	Sunny	Hot	High	Weak	Monday	No
2	Sunny	Cool	High	Strong	Tuesday	No
3	Overcast	Cool	Normal	Weak	Wednesday	No
4	Overcast	Hot	High	Weak	Thursday	Yes
5	Sunny	Mild	High	Weak	Friday	Yes
6	Rain	Cool	Normal	Weak	Saturday	Yes

In [16]:

```
import pandas as pd
from math import log2
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import norm
```

In [17]:

```
def calculate_entropy(data_frame, column):
    sum = 0
    current_unique = data_frame[column].unique()
    for i in current_unique:
        req1 = data_frame[column].value_counts()[i]/len(data_frame)
        req2 = log2(data_frame[column].value_counts()[i]/len(data_frame))
        sum+= -(req1)*req2
    return sum

def calculate_cond_entropy(current_column,data_frame,select_col):
    summed = 0
    collected_entropies = {}
    unique_vals = data_frame[current_column].unique()
    for i in unique_vals:
            val1 = (data_frame[current_column].value_counts()[i]/len(data_frame))
            val2 = calculate_entropy(data_frame[data_frame[current_column]==i],
                                     select_col)
            summed+= val1 * val2
            val3 = [calculate_entropy(data_frame[data_frame[current_column]==i],
                                      select_col)]
            collected_entropies[i] = val3
    return [summed, collected_entropies]

def calculate_information_gain(data_frame, current_column, select_col):
    req1 = calculate_entropy(data_frame, select_col)
    req2 = calculate_cond_entropy(current_column,data_frame,select_col)[0]
    req3 = calculate_cond_entropy(current_column,data_frame,select_col)[0]
    req4 = calculate_cond_entropy(current_column,data_frame,select_col)[1]
    return req1 - req2, req3,req4

def gini_calc(data_frame, col):
    sum =0
    unique_values = data_frame[col].unique()
    for i in unique_values:
        req1 = (data_frame[col].value_counts()[i]/len(data_frame))
        sum+= req1**2
    return 1 - sum

def gini_calc_(df, col, current_col):
    sum = 0
    unique_values = df[col].unique()
    for i in unique_values:
        req1 =  (df[col].value_counts()[i]/len(df))
        req2 = gini_calc(df[df[col]==i], current_col)
        sum+=req1*req2
    return sum
```

In [18]:

```
imported_values = pd.read_excel("hw.xlsx")
imported_values
```

Out[18]:

	Outlook	Temperature	Humidity	Wind	Day	Class
0	Sunny	Hot	High	Weak	Monday	No
1	Sunny	Cool	High	Strong	Tuesday	No
2	Overcast	Cool	Normal	Weak	Wednesday	No
3	Overcast	Hot	High	Weak	Thursday	Yes
4	Sunny	Mild	High	Weak	Friday	Yes
5	Rain	Cool	Normal	Weak	Saturday	Yes

In [19]:

```
calculate_entropy(imported_values, "Class")
```

Out[19]: 1.0

In [20]:

```
calculate_information_gain(imported_values, "Outlook", "Class")
```

Out[20]:

```
(0.20751874963942196,
 0.792481250360578,
 {'Sunny': [0.9182958340544896], 'Overcast': [1.0], 'Rain': [0.0]})
```

In [21]:

```
print(gini_calc_(imported_values, "Outlook", "Class"))
print(gini_calc_(imported_values, "Day", "Class"))
```

```
0.38888888888888884
0.0
```

In [22]:

```
calculate_information_gain(imported_values, "Day", "Class")
```

Out[22]:

```
(1.0,
 0.0,
 {'Monday': [0.0],
  'Tuesday': [0.0],
  'Wednesday': [0.0],
  'Thursday': [0.0],
  'Friday': [0.0],
  'Saturday': [0.0]})
```

Output class of the decision tree is the parent class. The entropy of the system is nothing but the entropy of parent. In this case, the parent is a class variable. Class Variable No 3 data points Yes 3 data points

1) The entropy of the system is 1

2) H(Y)=E(S)=1(Calculated in question 1)

If (Outlook=sunny & Class= YES), probability = 1/3

If (Outlook=sunny & Class= NO), probability = 2/3

H(Y==Sunny)=-1/3*log(1/3)* - -2/3 log(2/3)=0.91

If (Outlook=OVERCAST & Class= YES), probability = 1/2

If (Outlook=OVERCAST & Class= NO), probability = 1/2

H(Y==Overcast)=-1/2*log(1/2)* - -1/2 log(1/2)=1

If (Outlook=RAINY & Class= YES), probability = 1

If (Outlook=RAINY & Class= NO), probability = 0

H(Y==Rainy)=-1/1*log(1/1)* - -0/1 log(0/1)=0

Infogain(Outlook)=1-((3/6)*0.91+(2/6)*1+(1/6)*0)* =1-(0.5*0.91+0.33) = 0.215

3) P(Outlook=sunny): 3/6

P(Outlook=Overcast): 2/6

P(Outlook=Rainy): 1/6

If (Outlook=sunny & Class= YES), probability = 1/3

If (Outlook=sunny & Class= NO), probability = 2/3

Gini index = 1 - ((1/3)2 + (2/3)2) = 0.44

If (Outlook=OVERCAST & Class= YES), probability = 1/2

If (Outlook=OVERCAST & Class= NO), probability = 1/2

Gini index = 1 - ((1/2) + 2(1/2)2) = 0.5 If (Outlook=RAINY & Class= YES), probability = 1

If (Outlook=RAINY & Class= NO), probability = 0

Gini index = 1 - ((1)2 + (0)2) = 0

The weighted sum of the Gini Indices can be calculated as follows: Gini Index for Outlook = (3/6)*0.44 + (2/6)*0.5+(1/6)*0 = 0.385

4) The information gain"0" and Gini index is "1" for the 'day' attribute because all the instances of 'day' attribute are different so there is no dependency

5) Information gain of day attribute is zero. It implies that day attribute won't provide you with any information about the class attribute Remove entire feature in pre-processing or ignore while preparing decision tree

Question 4

Question 4 [4 pts]: Given six instances belonging to two classes: C_1={(1, 0, 1), (1, 1, 0), (0, -1, -1)} and C_2={(0, 1,-1), (-1, 0, 0), (-1, -1, 0)}. Each instance has three features. Assuming the class lable for C_1 and C_2 are 1 and 0, respectively. The learning rate =0.1, and the initial weights are w_0=1, w_1=1, w_2=1, and w_3=1 (w_0 is the weight value for the bias). Please use gradient descent learning rule to train a perceptron using above six training instances.

- The table below lists the six instances from C1 and C2 (the first dimension is the bias). Calculate the weight updating **W** for each individual instance for the first round using instance order given in the Table [2 pts] (show your calculation)
- Report weight values after the first round weight updating [1 pt].
- Report the mean squared errors E(W) after the weight updating for the first round [1 pt].

Input	Weight	v	Desired	Actual	W
(1,1,0,1)					
(1,1,1,0)					
(1,0,-1,-1)					
(1,0,1,-1)					
(1,-1,0,0)					
(1,-1,-1,0)					

In [43]:

```
import warnings
warnings.filterwarnings("ignore", category=np.VisibleDeprecationWarning)
def gradientDescent_question4(iteration, learning_rate, w):
    df = pd.DataFrame({"A1":[1, 1, 0, 0, -1, -1],
                "A2":[0, 1, -1, 1, 0, -1],
                "A3":[1, 0, -1, -1, 0, 0],
                 "y":[1, 1, 1, 0, 0, 0]})
    passed_in_features = df[["A1", "A2", "A3"]].values
    lab = df.y.values
    columns =  ["round", "inp", 'Weight', 'v', 'Desired', 'lr(desire(n)-ooutput(n))',
             '$\Delta$ Wi', '$\Delta$ W', "Error", "W(k+1)"]
    results = pd.DataFrame(columns = columns)
    current_index=1
    for round in range(iteration):
        w_=w
        accuracy_delta = np.zeros(len(w))
        errors = []
        for i, value in enumerate(passed_in_features):
            value = np.insert(value,0,1)
            v=np.dot(w, value.transpose())
            output = learning_rate*(lab[i]-v)
            errors.append( lab[i]-v)
            deltaw = output*value
            accuracy_delta += deltaw
            cols = ["round", "inp", 'Weight', 'v', 'Desired', 'lr(desire(n)-ooutput(n))', '$\Delta$ Wi', '$\Del
            results.loc[current_index,cols] = [round+1, value, ["%2.2f"%k for k in w_],
                                v, lab[i],
                                output, ["%2.1f"%k for k in deltaw],
                                np.array(["%2.1f"%k for k in accuracy_delta])]
            current_index+=1
        w = w + accuracy_delta
        equation = (np.array(errors)**2.0).sum()/len(passed_in_features)
        results.loc[current_index-1, ["Error", "W(k+1)"]] = [equation, ["%2.1f"%k for k in w]]
    return results


gradientDescent_question4(2, 0.1, np.array([1, 1, 1, 1]))

#req1 = (desired - v)**2
#reqq2 = req1.sum()
#req3 = reqq2/6
#After Error(We) = req3  = 1.8
#req1 = (desired - value)**2
#reqq2 = req1.sum()
#req3 = reqq2/2*6
#mean squared error after (1/n) = req3  = 0.3
```

Out[43]:

	round	inp	Weight	v	Desired	lr(desire(n)-ooutput(n))	Δ Wi	Δ W	Error	W(k+1)
1	1	[1, 1, 0, 1]	[1.00, 1.00, 1.00, 1.00]	3	1	-0.2	[-0.2, -0.2, -0.0, -0.2]	[-0.2, -0.2, 0.0, -0.2]	NaN	NaN
2	1	[1, 1, 1, 0]	[1.00, 1.00, 1.00, 1.00]	3	1	-0.2	[-0.2, -0.2, -0.2, -0.0]	[-0.4, -0.4, -0.2, -0.2]	NaN	NaN
3	1	[1, 0, -1, -1]	[1.00, 1.00, 1.00, 1.00]	-1	1	0.2	[0.2, 0.0, -0.2, -0.2]	[-0.2, -0.4, -0.4, -0.4]	NaN	NaN
4	1	[1, 0, 1, -1]	[1.00, 1.00, 1.00, 1.00]	1	0	-0.1	[-0.1, -0.0, -0.1, 0.1]	[-0.3, -0.4, -0.5, -0.3]	NaN	NaN
5	1	[1, -1, 0, 0]	[1.00, 1.00, 1.00, 1.00]	0	0	0.0	[0.0, -0.0, 0.0, 0.0]	[-0.3, -0.4, -0.5, -0.3]	NaN	NaN
6	1	[1, -1, -1, 0]	[1.00, 1.00, 1.00, 1.00]	-1	0	0.1	[0.1, -0.1, -0.1, 0.0]	[-0.2, -0.5, -0.6, -0.3]	2.333333	[0.8, 0.5, 0.4, 0.7]
7	2	[1, 1, 0, 1]	[0.80, 0.50, 0.40, 0.70]	2.0	1	-0.1	[-0.1, -0.1, -0.0, -0.1]	[-0.1, -0.1, 0.0, -0.1]	NaN	NaN
8	2	[1, 1, 1, 0]	[0.80, 0.50, 0.40, 0.70]	1.7	1	-0.07	[-0.1, -0.1, -0.1, -0.0]	[-0.2, -0.2, -0.1, -0.1]	NaN	NaN
9	2	[1, 0, -1, -1]	[0.80, 0.50, 0.40, 0.70]	-0.3	1	0.13	[0.1, 0.0, -0.1, -0.1]	[-0.0, -0.2, -0.2, -0.2]	NaN	NaN
10	2	[1, 0, 1, -1]	[0.80, 0.50, 0.40, 0.70]	0.5	0	-0.05	[-0.1, -0.0, -0.1, 0.1]	[-0.1, -0.2, -0.2, -0.2]	NaN	NaN
11	2	[1, -1, 0, 0]	[0.80, 0.50, 0.40, 0.70]	0.3	0	-0.03	[-0.0, 0.0, -0.0, -0.0]	[-0.1, -0.1, -0.2, -0.2]	NaN	NaN
12	2	[1, -1, -1, 0]	[0.80, 0.50, 0.40, 0.70]	-0.1	0	0.01	[0.0, -0.0, -0.0, 0.0]	[-0.1, -0.1, -0.3, -0.2]	0.588333	[0.7, 0.4, 0.1, 0.5]

Question 5 [4 pts]: Given four instances belonging to two classes: C_1={(1, 0, 1), (1, 1, 0) } and C_2={(0, 1,-1), (-1, 0, 0)}. Each instance has three features. Assuming the class lable for C_1 and C_2 are 1 and 0, respectively. The learning rate =0.1, and the initial weights are w_0=1, w_1=1, w_2=1, and w_3=1 (w_0 is the weight value for the bias). Please use Delta rule (AdaLine: Adaptive Linear Elements) to train a perceptron for these two classes.

- The table below lists the six instances from C1 and C2 (the first dimension is the bias). Calculate the results of the first round weight updating using instance order listed in the Table. Show final weight values after the first round weight updating [2 pts]
- Calculate the network mean squared error BEFORE the weight updating, and AFTER the weight updating for the last instance, respectively [2 pts]

Input	Weight	v	Desired	Actual	W	New Weight
(1,1,0,1)						
(1,1,1,0)						
(1,0,1,-1)						
(1,-1,0,0)						

In [40]:

```
def delta_question_5(num_iter, learning_rate, w):
    df = pd.DataFrame({"X1":[1, 1, 0, -1],
                "X2":[0, 1, 1, 0],
                "X3":[1, 0, -1, 0],
                 "y":[1, 1, 0, 0,]})
    features = df[["X1", "X2", "X3"]].values
    labels = df.y.values
    cols = ["Round", "inp", 'Weight', 'd__', 'O', 'learn_rate(desire-output)$X_i$', 'New Weight', "Error"]
    results = pd.DataFrame(columns = cols)
    j = 0
    error = 0
    for round in range(num_iter):
        for i, x in enumerate(features):
            x = np.insert(x, 0, 1)
            v= np.dot(w, x.transpose())
            current_delta = learning_rate * (labels[i]-v) * x
            error = error + (labels[i]-v)**2
            w_ = w
            w = w + current_delta
            j+=1
            cols = ["Round", "inp", 'Weight', 'd__', 'O', 'learn_rate(desire-output)$X_i$', 'New Weight']
            results.loc[j,cols] = [round+1,
                                   x,
                                   np.array(["%2.1f"%k for k in w_ ]),
                                   labels[i],
                                   v,
                                   ["%2.1f"%k for k in current_delta ],
                                   ["%.1f"%k for k in w ]]

            if (j%len(features)==0):
                results.loc[j,"Error"] = error/len(features)

    return results
delta_question_5(num_iter=1, learning_rate=0.1, w = np.array([1, 1, 1,1]))

#Before
# req1 = delta_rate.error**2
# req2 = req1.saum()
# req3 = req2/2
# E(W) =  req3 = 4.5557

#req1 = delta_rate.Error**2
#req2 = req1.sum()
#req3 = req2/(4*2)
# Mean square error = req3 = 1.3


# After
#req1 = (deltarate.delta- deltarate.Ouput)**2
#req2 = req1.sum()
#req3 = req2/2
# E(W) =  req3 = 3.566 $
# val = (2*4)
# val2 = (deltarate.delta- deltarate.output)**2).sum()
# Mean square error(1/n) = (val2/val) = 0.888 $
```

Out[40]:

	Round	inp	Weight	d__	O	learn_rate(desire-output)X_i	New Weight	Error
1	1	[1, 1, 0, 1]	[1.0, 1.0, 1.0, 1.0]	1	3	[-0.2, -0.2, -0.0, -0.2]	[0.8, 0.8, 1.0, 0.8]	NaN
2	1	[1, 1, 1, 0]	[0.8, 0.8, 1.0, 0.8]	1	2.6	[-0.2, -0.2, -0.2, -0.0]	[0.6, 0.6, 0.8, 0.8]	NaN
3	1	[1, 0, 1, -1]	[0.6, 0.6, 0.8, 0.8]	0	0.68	[-0.1, -0.0, -0.1, 0.1]	[0.6, 0.6, 0.8, 0.9]	NaN
4	1	[1, -1, 0, 0]	[0.6, 0.6, 0.8, 0.9]	0	-0.068	[0.0, -0.0, 0.0, 0.0]	[0.6, 0.6, 0.8, 0.9]	1.756756

Question 6 [3 pts]: Assume you were asked to design an RBF network to classify instances in three dimensional space (i.e., each instance has three features). The basis function (r) of the RBF network is defined as $(r)=r^2$. The network has two centers, and the corresponding centers of the basis function are $t_1=[1, 0, -1]^T$ and $t_2=[1, 1, 0]^T$, respectively. Assume the bias of the network is 1, and the weight values of the RBF network are [1, -1, 2] (where 1 is the weight value of the bias).

- Draw an RBF network corresponding to the above settings [1 pt]
- Given an input instance $x=[-1, 1, 2]^T$, please computer the output of the instance x from the RBF network [2 pts] (please show your solutions).

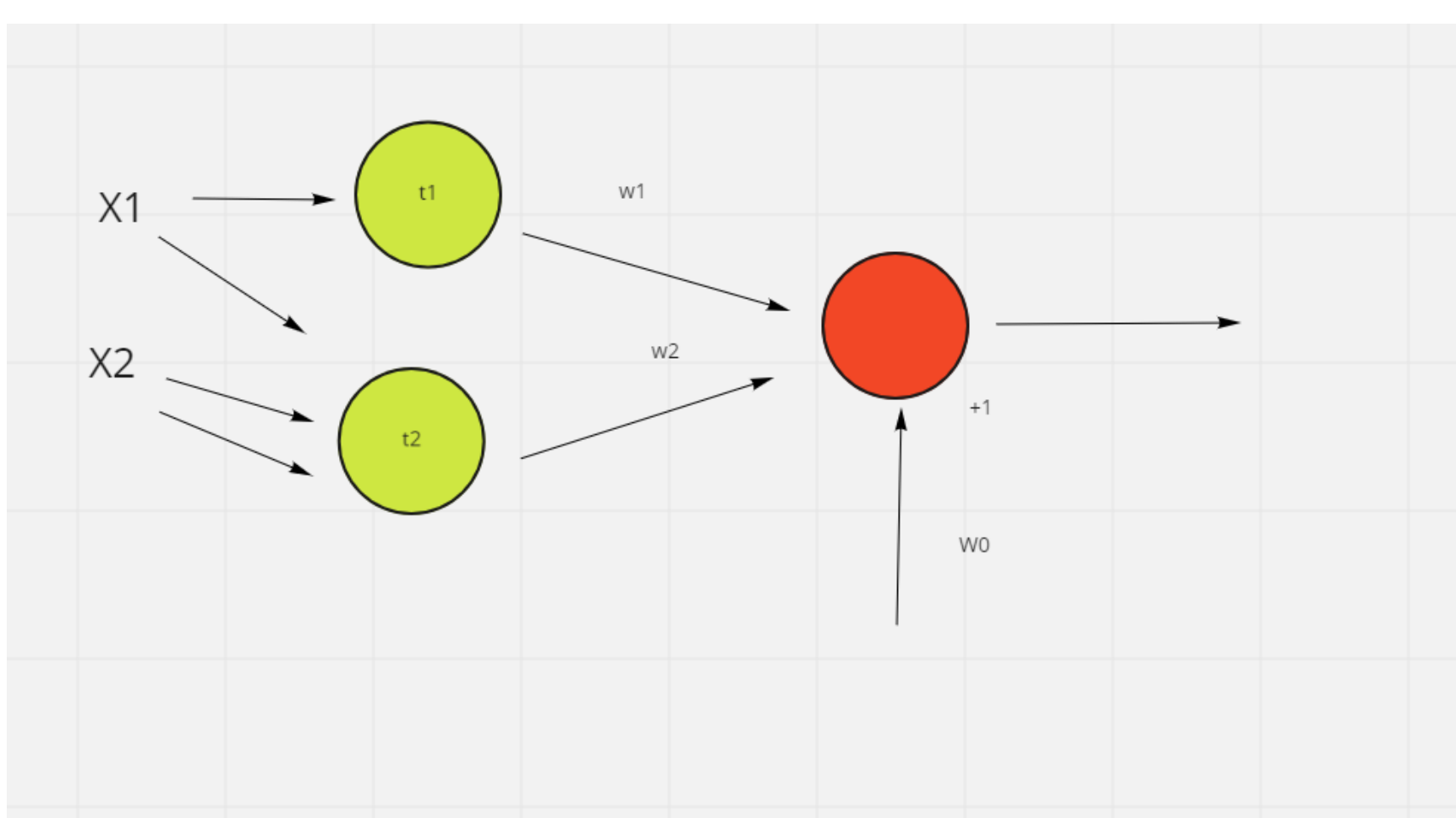
In [25]:

```
def question6_rbf():
    requirement_inst = pd.DataFrame(
        {"A1":[-1],
         "A2":[1],
        "A3":[2],}
    )
    t1 = np.array([1, 0, -1])
    t2 = np.array([1, 1, 0])
    display(requirement_inst)
    sq = lambda x1, x2,x3, t: norm(np.array([x1, x2, x3]) - t)**2
    data_frame_instance = requirement_inst.iloc[:, :]
    data_frame_instance["z1"] = data_frame_instance.apply(lambda x: sq(x["A1"], x["A2"],x["A3"], t1), axis=1)
    data_frame_instance["z2"] = data_frame_instance.apply(lambda x: sq(x["A1"], x["A2"],x["A3"], t2), axis=1)
    display(data_frame_instance)
    data_frame_instance["Verify"] = data_frame_instance.apply(lambda x: 1 + -1*x["z1"] +2*x["z2"] , axis = 1)
    display(data_frame_instance)
question6_rbf()
```

	A1	A2	A3
0	-1	1	2

	A1	A2	A3	z1	z2
0	-1	1	2	14.0	8.0

	A1	A2	A3	z1	z2	Verify
0	-1	1	2	14.0	8.0	3.0

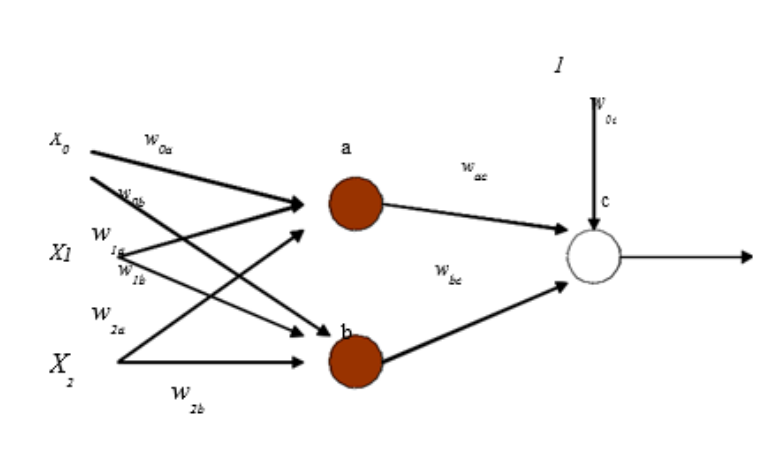


Question 7 [5 pts]: Given the following multi-layer feedforward neural network with one hidden layer and one output layer, assuming the network initial weights are w_{0a} = 0.5, w_{1a} = 0.5; w_{2a} = 0.5; w_{0b} = 1.0, w_{1b} = 1.0; w_{2b} = 1.0; w_{0c} = 0.5, w_{ac} = 0.5, w_{bc} = 0.5. All nodes use a sigmoid activation function with value a=1.0, and the learning rate is set to 0.5. The expected output is *1* if an instance's class label is "True", otherwise, the expected output is *0*.

Given a training instance I_1, which is labeled as "True", please calculate its actual output from the network [1 pt], and the mean squared error of the network with respect to the instance [1 pt]

Assuming instance I_1 is fed to the network to train the network weight values, please update the network weight (using Backpropogation rule) by using this instance (list major steps and results) [3 pts]

I_1 = [1.0, 1.0, 0.5]



Calculation of Oa, Ob, Oc for the instance given I1

Oa(I1) = Φ (1.0+1.0+0.5) =1/(1+e^-1*2.5) = 0.9241

Ob(I1) = Φ(1.0+1.0+0.5) = 1/(1+e^-1*2.5)=0.9241

now for Oc

Oc(I1) = Φ(0.5+0.5Oa(I1) +0.5Ob(I1) =0.62036

what is the mean squared error of the network over the instance given I1

I1: o.5(da-Oa)2=0.5(0.62036)2=0.19242

The total squared error E(W)= 0.1924

Mean Squared Error E(W)/Total number of instance

= 0.19242/1

MSE E(W)/1 = 0.19242 I, instance used is fed to the network to update the weight

Calculate local gradient of each node

δ_c = Oc(1 – Oc) * (dc – Oc)

δ_b = Ob(1 – Ob) □(*Wbc* □c)

δ_a = Oa(1 – Oa) □(*Wac* □c)

δ_c = 0.62036(1-0.62036)*(0-0.62036) = -0.1461

δ_b = 0.9241(1-0.9241)*(0.5*-0.1461) =0.005123

δ_a =0.9241(1-0.9241)*(0.5*-0.1461) =0.005123

Δ

Update the network weight using backpropagation rule

Δ Wc=[ΔWOc, Δ Wac, Δ Wbc]=E Δ *j*Xij=-0.07305

WOc 0.5 + -0.07305 = -0.42695

Wac 0.5 + -0.07305 = -0.42695

Wbc 0.5 + -0.07305 = -0.42695

For B Δ W=EΔ *j*Xij= 0.0025615

Wob 1 + 0.0025615 = 0.0025615

W1b 1 + 0.0025615 = 0.0025615

W2b 1 + 0.0025615 = 0.0025615

For A Δ W=EΔ*j*Xij= 0.0025615

Woa 0.5 + 0.0025615 = 0.502515

W1a 0.5 + 0.0025615 = 0.502515

W2a 05 + 0.0025615 = 0.502515

Question 8 [2 pts]:

a. What is the most difficult part of this class to you (theories, mathematics, programming, homework, logistics, communication, time, etc.) [20 words minimum]?

I think the most difficult part was watching all the videos so many videos and so little time. Alot of information to ingest. I have two kids and I work two jobs so I am up super late watching all videos and catching up ppt. This is probably the most challenging class i have taken.

b. Would you recommend any changes to the course [30 words minimum]? I think the best thing to do would be shorten videos and give more ocntent thats core to what we need so i can read a pdf instead of trying to extract tribal knowledge from videos. I know alot of the stuff is in the ppt's but i found myself missing some key info from videos.

c. In a scale of 1 to 10 (1 being the least, and 10 being the most), how much you have learned from this class? [a numeric number, and any explanations if necessary] I think i learned at a high 9 alot of the stuff in this class helps build conceptual knowledge and skills at the same time to use in the workforce. Which is exactly what i was looking for. I just wish i had more time in this class. I can soak up more of the information.

In [ ]: