# Neural Networks

Homework 4
By:
Monil Shah
mds747
N19510262

In [80]:
```
import numpy as np
```

In [81]:
```
X = np.asarray([[1, 1], [1, 2], [2, -1], [2, 0], [-1, 2], [-2, 1], [-1, -1],
y=  np.asarray([[-1, -1], [-1, -1], [-1, 1], [-1, 1], [1, -1], [1, -1], [1,
```

# Delta Rule

In [82]:
```
def hardlim(y_in,train=False):
    if train:
        return y_in
    else:
            y_out = [[],[]]
            for x in range(len(y_in)):
                if y_in[x] >= 0:
                    y_out[x] = 1
                if y_in[x] < 0:
                    y_out[x] = -1
            return y_out

def yin(x_vec):
    y_in = np.zeros(bias.shape) + bias
    for j in range(len(bias)):
        y_in[j] += np.dot(x_vec, w[j])
    return y_in
```

# Alpha = 0.001

In [83]:
```
w=np.zeros(shape=(X.shape[1], y.shape[1]))
bias=np.zeros(shape=y.shape[1])
alpha=0.001
y = y.reshape(y.shape[0],-1)
if y.shape[1] == 1:
    w = w.reshape(1, w.shape[0])
    bias = np.reshape(bias, -1)
for i in range(1,10000):
    for x_vec, y_vec in zip(X, y):
        y_in = yin(x_vec)
        yy = []
        for j in range(len(bias)):
            yy.append(hardlim(y_in[j],train=True))
            w[j] = w[j] - (2*alpha*np.outer(np.subtract(yy[j],y_vec[j]),x_ve
            bias[j] = bias[j] - (2*alpha*np.subtract(yy[j],y_vec[j]))
```

In [84]:
```
print("Weights: \n",w.transpose())
print("Bias: \n",bias)
```

```
Weights:
 [[-0.59367939  0.16801311]
 [-0.05095887 -0.6659139 ]]
Bias:
 [ 0.01325352  0.1679855 ]
```

In [101]:
```
count = 0
target = []
for i in X:
    yy = i.dot(w) + bias
    y_out = hardlim(yy)
    target.append(y_out)
print("Predicted Output: ",target)
```

```
Predicted Output:  [[-1, -1], [-1, -1], [-1, 1], [-1, 1], [1, -1], [1, -
1], [1, 1], [1, 1]]
```

In [114]:
```
if((y==target).all()):
    print("All inputs classified correctly")
```

```
All inputs classified correctly
```

# Alpha = 0.1

In [87]:
```
w=np.zeros(shape=(X.shape[1], y.shape[1]))
bias=np.zeros(shape=y.shape[1])
alpha=0.1
y = y.reshape(y.shape[0],-1)
if y.shape[1] == 1:
    w = w.reshape(1, w.shape[0])
    bias = np.reshape(bias, -1)
for i in range(1,10000):
    for x_vec, y_vec in zip(X, y):
        y_in = yin(x_vec)
        yy = []
        for j in range(len(bias)):
            yy.append(hardlim(y_in[j],train=True))
            w[j] = w[j] - (2*alpha*np.outer(np.subtract(yy[j],y_vec[j]),x_ve
            bias[j] = bias[j] - (2*alpha*np.subtract(yy[j],y_vec[j]))
```

In [88]:
```
print("Weights: \n",w.transpose())
print("Bias: \n",bias)
```

```
Weights:
 [[-0.39953017  0.35521879]
 [ 0.06679871 -0.60212805]]
Bias:
 [ 0.06760018  0.38030401]
```

In [102]:
```
count = 0
target = []
for i in X:
    yy = i.dot(w) + bias
    y_out = hardlim(yy)
    target.append(y_out)
print("Predicted Output: ",target)
```

```
Predicted Output:  [[-1, -1], [-1, -1], [-1, 1], [-1, 1], [1, -1], [1, -
1], [1, 1], [1, 1]]
```

In [105]:
```
if((y==target).all()):
    print("All inputs classified correctly")
```

```
All inputs classified correctly
```

# Part 2: Gradually Decreasing alpha

# Alpha = 1/Iteration

In [91]:
```
w=np.zeros(shape=(X.shape[1], y.shape[1]))
bias=np.zeros(shape=y.shape[1])
y = y.reshape(y.shape[0],-1)
if y.shape[1] == 1:
    w = w.reshape(1, w.shape[0])
    bias = np.reshape(bias, -1)
for i in range(1,10000):
    for x_vec, y_vec in zip(X, y):
        alpha=1/i;
        y_in = yin(x_vec)
        yy = []
        for j in range(len(bias)):
            yy.append(hardlim(y_in[j],train=True))
            w[j] = w[j] - (2*alpha*np.outer(np.subtract(yy[j],y_vec[j]),x_ve
            bias[j] = bias[j] - (2*alpha*np.subtract(yy[j],y_vec[j]))
```

In [97]:
```
print("Weights: \n",w.transpose())
print("Bias: \n",bias)
```

```
Weights: 
 [[-0.59466015  0.16680423]
 [-0.0521512  -0.66659059]]
Bias: 
 [ 0.01308995  0.16680646]
```

In [103]:
```
count = 0
target = []
for i in X:
    yy = i.dot(w) + bias
    y_out = hardlim(yy)
    target.append(y_out)
print("Predicted Output: ",target)
```

```
Predicted Output:  [[-1, -1], [-1, -1], [-1, 1], [-1, 1], [1, -1], [1, -
1], [1, 1], [1, 1]]
```

In [104]:
```
if((y==target).all()):
    print("All inputs classified correctly")
```

```
All inputs classified correctly
```

# Alpha = alpha*0.9

In [111]:
```
alpha=0.1
w=np.zeros(shape=(X.shape[1], y.shape[1]))
bias=np.zeros(shape=y.shape[1])
y = y.reshape(y.shape[0],-1)
if y.shape[1] == 1:
    w = w.reshape(1, w.shape[0])
    bias = np.reshape(bias, -1)
for i in range(1,10000):
    for x_vec, y_vec in zip(X, y):
        alpha=alpha*0.9;
        y_in = yin(x_vec)
        yy = []
        for j in range(len(bias)):
            yy.append(hardlim(y_in[j],train=True))
            w[j] = w[j] - (2*alpha*np.outer(np.subtract(yy[j],y_vec[j]),x_ve
            bias[j] = bias[j] - (2*alpha*np.subtract(yy[j],y_vec[j]))
print("Weights: \n",w.transpose())
print("Bias: \n",bias)
```

```
Weights: 
 [[-0.59392236  0.17147829]
 [-0.02947102 -0.64811857]]
Bias: 
 [-0.04044308  0.16911988]
```

In [112]:
```
count = 0
target = []
for i in X:
    yy = i.dot(w) + bias
    y_out = hardlim(yy)
    target.append(y_out)
print("Predicted Output: ",target)
```

```
Predicted Output:  [[-1, -1], [-1, -1], [-1, 1], [-1, 1], [1, -1], [1, -
1], [1, 1], [1, 1]]
```

In [113]:
```
if((y==target).all()):
    print("All inputs classified correctly")
```

```
All inputs classified correctly
```

Comment on the results:
As we can see , we get better results for weight and bais, hence we get less error with value of alpha nearer to zero.

In [ ]:
```
```