M.Nilofer Sultana,

AP17110010058,

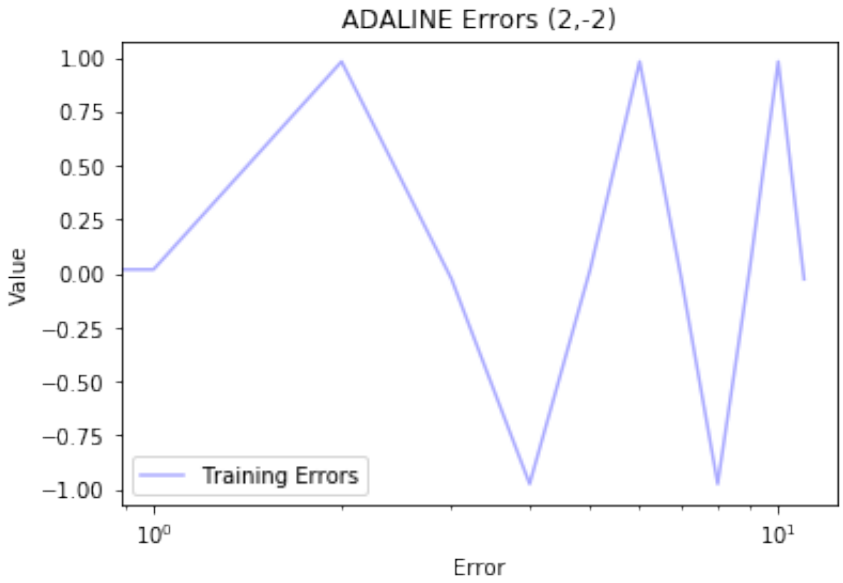
CSE-01.

Implementing OR using Adaline

In [246]:

```
import matplotlib.pyplot as plt
import numpy as np
b=0.45
x0=[1,1,1,1]
x1=[1,1,-1,-1]
x2=[1,-1,1,-1]
t =[1,1,1,-1]
w = 2*np.random.random((3,1)) - 1
print(w)
y=[0,0,0,0]
y_out=[0,0,0,0]
er=[]
for j in range (3):
    print("epoch",j+1)
    for i in range(4):
        x=(x0[i]*w[0])+(x1[i]*w[1])+(x2[i]*w[2])
        y[i]=x
        if (y[i]>=0):
            y_out[i]=1
        else:
            y_out[i]=-1
        dif=t[i]-x
        er.append(dif)
        if (y_out[i]!=t[i]):
            w[0]=w[0]+(b*dif*x0[i])
            w[1]=w[1]+(b*dif*x1[i])
            w[2]=w[2]+(b*dif*x2[i])
            print(w)
    for i in range(4):
        x=(x0[i]*w[0])+(x1[i]*w[1])+(x2[i]*w[2])
        y[i]=x
        if (x>0):
            y_out[i]=1
        else:
            y_out[i]=-1
    print("Acutal ",y_out,"Desired  ",t)
ax = plt.subplot(111)
ax.plot(er, c="#aaaaff", label="Training Errors")
ax.set_xscale("log")
plt.title("ADALINE Errors (2,-2)")
plt.legend()
plt.xlabel("Error")
plt.ylabel("Value")
plt.show()
```

```
[[0.50028863]
 [0.97772218]
 [0.49633131]]
epoch 1
Acutal  [1, 1, 1, -1] Desired   [1, 1, 1, -1]
epoch 2
Acutal  [1, 1, 1, -1] Desired   [1, 1, 1, -1]
epoch 3
Acutal  [1, 1, 1, -1] Desired   [1, 1, 1, -1]
```

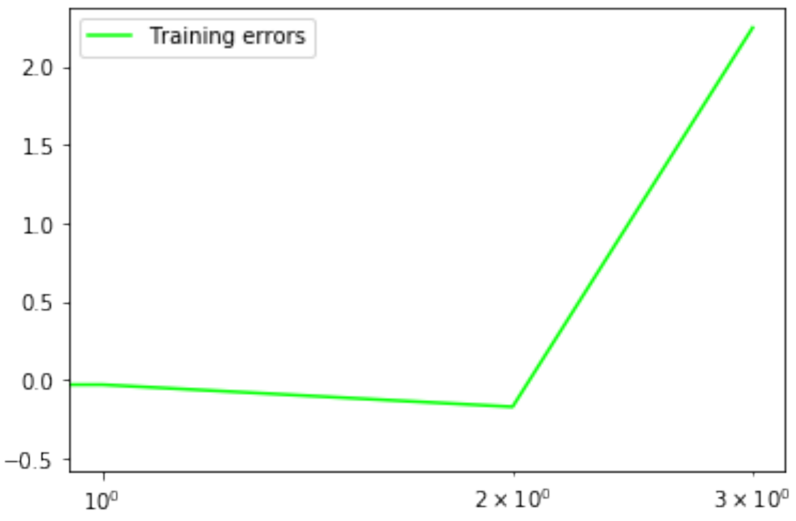


Implementing AND using Adaline

In [2]:

```
import matplotlib.pyplot as plt
import numpy as np
b=0.45
x0=[1,1,1,1]
x1=[1,1,-1,-1]
x2=[1,-1,1,-1]
t =[1,-1,-1,-1]
w = 2*np.random.random((3,1)) - 1
print(w)
y=[0,0,0,0]
y_out=[0,0,0,0]
er=[None]*5
a=[None]*5
for j in range (5):
    print("epoch",j+1)
    for i in range(4):
        x=(x0[i]*w[0])+(x1[i]*w[1])+(x2[i]*w[2])
        y[i]=x
        if (y[i]>=0):
            y_out[i]=1
        else:
            y_out[i]=-1
        dif=t[i]-x
        er[i]=dif
        if (y_out[i]!=t[i]):
            w[0]=w[0]+(b*dif*x0[i])
            w[1]=w[1]+(b*dif*x1[i])
            w[2]=w[2]+(b*dif*x2[i])
            print(w)
    for i in range(4):
        x=(x0[i]*w[0])+(x1[i]*w[1])+(x2[i]*w[2])
        y[i]=x
        if (x>0):
            y_out[i]=1
        else:
            y_out[i]=-1
    print("Acutal ",y_out,"Desired  ",t)
ax = plt.subplot(111)
ax.plot(er,c="#00ff00",label='Training errors')
ax.set_xscale("log")
plt.legend()
plt.show()
```

```
[[ 0.90831597]
 [-0.66003107]
 [ 0.29098966]]
epoch 1
[[-0.37838554]
 [ 0.62667045]
 [-0.99571186]]
Acutal  [-1, 1, -1, -1] Desired   [1, -1, -1, -1]
epoch 2
[[ 0.40795659]
 [ 1.41301258]
 [-0.20936973]]
[[-0.95569591]
 [ 0.04936007]
 [ 1.15428277]]
[[-1.47284797]
 [ 0.56651213]
 [ 0.63713072]]
Acutal  [-1, -1, -1, -1] Desired   [1, -1, -1, -1]
epoch 3
[[-0.90170566]
 [ 1.13765443]
 [ 1.20827302]]
Acutal  [1, -1, -1, -1] Desired   [1, -1, -1, -1]
epoch 4
Acutal  [1, -1, -1, -1] Desired   [1, -1, -1, -1]
epoch 5
Acutal  [1, -1, -1, -1] Desired   [1, -1, -1, -1]
```

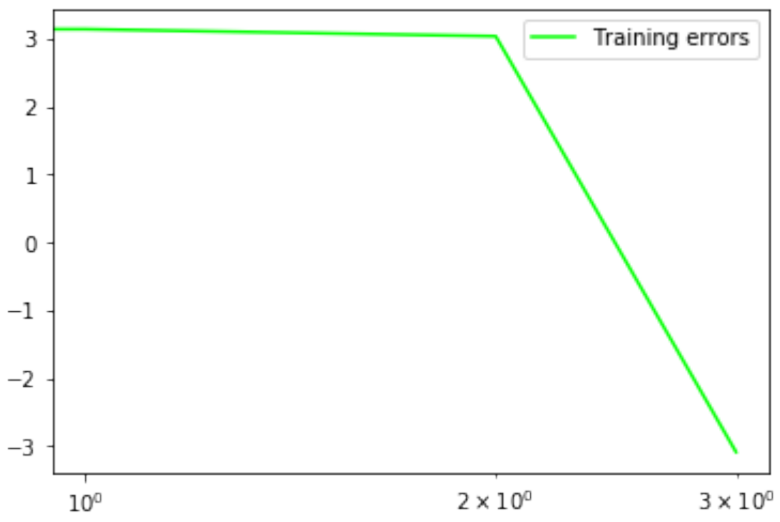


Implementing XOR using Adaline

```
In [3]: import matplotlib.pyplot as plt
        import numpy as np
        b=0.45
        x0=[1,1,1,1]
        x1=[1,1,-1,-1]
        x2=[1,-1,1,-1]
        t =[-1,1,1,-1]
        w = 2*np.random.random((3,1)) - 1
        print(w)
        y=[0,0,0,0]
        y_out=[0,0,0,0]
        er=[None]*5
        a=[None]*5
        for j in range (5):
            print("epoch",j+1)
            for i in range(4):
                x=(x0[i]*w[0])+(x1[i]*w[1])+(x2[i]*w[2])
                y[i]=x
                if (y[i]>=0):
                    y_out[i]=1
                else:
                    y_out[i]=-1
                dif=t[i]-x
                er[i]=dif
                if (y_out[i]!=t[i]):
                    w[0]=w[0]+(b*dif*x0[i])
                    w[1]=w[1]+(b*dif*x1[i])
                    w[2]=w[2]+(b*dif*x2[i])
                    print(w)
            for i in range(4):
                x=(x0[i]*w[0])+(x1[i]*w[1])+(x2[i]*w[2])
                y[i]=x
                if (x>0):
                    y_out[i]=1
                else:
```

```
    else:
      y_out[i]=-1
  print("Acutal ",y_out,"Desired  ",t)
ax = plt.subplot(111)
ax.plot(er,c="#00ff00",label='Training errors')
ax.set_xscale("log")
plt.legend()
plt.show()
```

```
[[ 0.97281281]
 [-0.63229226]
 [-0.55900287]]
epoch 1
[[-0.45103576]
 [ 0.79155632]
 [ 0.8648457 ]]
Acutal  [1, -1, -1, -1] Desired   [-1, 1, 1, -1]
epoch 2
[[-1.44345058]
 [-0.2008585 ]
 [-0.12756912]]
[[-0.31091759]
 [ 0.93167448]
 [-1.2601021 ]]
[[ 1.26529478]
 [-0.6445379 ]
 [ 0.31611028]]
[[0.0981197 ]
 [0.52263718]
 [1.48328536]]
Acutal  [1, -1, 1, -1] Desired   [-1, 1, 1, -1]
epoch 3
[[-1.29869931]
 [-0.87418183]
 [ 0.08646635]]
[[ 0.16800706]
 [ 0.59252454]
 [-1.38024002]]
[[ 1.43014793]
 [-0.66961633]
 [-0.11809915]]
[[-0.0178906 ]
 [ 0.7784222 ]
 [ 1.32993939]]
Acutal  [1, -1, 1, -1] Desired   [-1, 1, 1, -1]
epoch 4
[[-1.40860255]
 [-0.61228974]
 [-0.06077255]]
[[-0.07654867]
 [ 0.71976414]
 [-1.39282644]]
[[ 1.35856399]
 [-0.71534852]
 [ 0.04228622]]
[[-0.00566784]
 [ 0.64888331]
 [ 1.40651805]]
Acutal  [1, -1, 1, -1] Desired   [-1, 1, 1, -1]
epoch 5
[[-1.37804792]
 [-0.72349678]
 [ 0.03413797]]
[[ 0.03300928]
 [ 0.68756042]
 [-1.37691923]]
[[ 1.39717095]
 [-0.67660125]
 [-0.01275756]]
[[0.00823256]
 [0.71233714]
 [1.37618083]]
Acutal  [1, -1, 1, -1] Desired   [-1, 1, 1, -1]
```

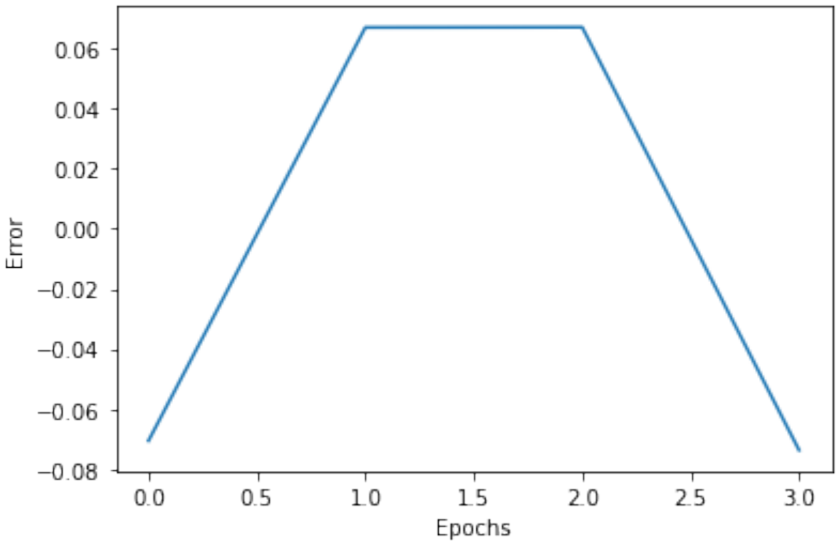


Implementing XOR using Madaline

In [23]:

```
import numpy as np
import matplotlib.pyplot as plt
#np.random.seed(0)

def sigmoid (x):
    return 1/(1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

#Input datasets
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
target = np.array([[0],[1],[1],[0]])

epochs = 10000
epochs = 10000
lr = 0.1
er=[]
inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons = 2,2,1

#Random weights and bias initialization
hidden_weights = np.random.uniform(size=(inputLayerNeurons,hiddenLa
hidden_bias =np.random.uniform(size=(1,hiddenLayerNeurons))
output_weights = np.random.uniform(size=(hiddenLayerNeurons,outputL
output_bias = np.random.uniform(size=(1,outputLayerNeurons))

print("Initial hidden weights: ",end='')
print(*hidden_weights)
print("Initial hidden biases: ",end='')
print(*hidden_bias)
print("Initial output weights: ",end='')
print(*output_weights)
print("Initial output biases: ",end='')
print(*output_bias)


#Training algorithm
for _ in range(epochs):
#Forward Propagation
    hidden_layer_activation = np.dot(inputs,hidden_weights)
    hidden_layer_activation += hidden_bias
    hidden_layer_output = sigmoid(hidden_layer_activation)

    output_layer_activation = np.dot(hidden_layer_output,output_wei
    output_layer_activation += output_bias
    predicted_output = sigmoid(output_layer_activation)

#Backpropagation
    error = target - predicted_output
    er.append(error)
    d_predicted_output = error * sigmoid_derivative(predicted_outpu
    error_hidden_layer = d_predicted_output.dot(output_weights.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden

#Updating Weights and Biases
    output_weights += hidden_layer_output.T.dot(d_predicted_output)
    output_bias += np.sum(d_predicted_output,axis=0,keepdims=True)
    hidden_weights += inputs.T.dot(d_hidden_layer) * lr
    hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) * lr

print("Final hidden weights: ",end='')
print(*hidden_weights)
print("Final hidden bias: ",end='')
print(*hidden_bias)
print("Final output weights: ",end='')
print(*output_weights)
print("Final output bias: ",end='')
print(*output_bias)

print("\nOutput from neural network after 10,000 epochs: ",end='')
print("\nOutput from neural network after 10,000 epochs: ",end=" ")
print(*predicted_output)
ax = plt.subplot(111)
x_line=np.linspace(0,1)
plt.plot(error)
plt.xlabel("Epochs")
plt.ylabel("Error")
plt.show()
```

```
Initial hidden weights: [0.02286355 0.86046719] [0.283145   0.5667
9159]
Initial hidden biases: [0.66192064 0.77672574]
Initial output weights: [0.7822206] [0.09087432]
Initial output biases: [0.48715264]
Final hidden weights: [3.48503873 5.69065929] [3.4885271  5.709948
55]
Final hidden bias: [-5.32273513 -2.31108219]
Final output weights: [-7.69490057] [7.10716369]
Final output bias: [-3.18254136]

Output from neural network after 10,000 epochs: [0.0705266] [0.933
06776] [0.93299628] [0.07364187]
```



Feed forward error back propagation learning for iris dataset

In [3]:
```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
from sklearn import datasets
iris = datasets.load_iris()
iris.loc[iris["Name"]=="virginica","species"]=0
iris.loc[iris["Name"]=="versicolor","species"]=1
iris.loc[iris["Name"]=="setosa","species"] = 2
iris = iris[iris["species"]!=2]
X = iris[["PetalLength", "PetalWidth"]].values.T
Y = iris[["species"]].values.T
```

```
Y = iris[['species']].values.T
Y = Y.astype("uint8")
plt.scatter(X[0, :], X[1, :], c=Y[0,:], s=40, cmap=plt.cm.Spectral)
plt.title("IRIS DATA | Blue - Versicolor, Red - Virginica ")
plt.xlabel("Petal Length")
plt.ylabel("Petal Width")
plt.show()
def old_para(n_x, n_h, n_y):
    np.random.seed(2)
    W1 = np.random.randn(n_h, n_x) * 0.01
    b1 = np.zeros(shape=(n_h, 1))
    W2 = np.random.randn(n_y, n_h) * 0.01
    b2 = np.zeros(shape=(n_y, 1))
    parameters = {"W1": W1,
    "b1": b1,
    "W2": W2,
    "b2": b2}
    return parameters
def size(X, Y):
    n_x = X.shape[0]
    n_h = 6
    n_y = Y.shape[0]
    return (n_x, n_h, n_y)
def forward_propagation(X, parameters):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    Z1 = np.dot(W1, X) + b1
    A1 = np.tanh(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = 1/(1+np.exp(-Z2))
    cache = {"Z1": Z1,
    "A1": A1,
    "Z2": Z2,
    "A2": A2}
    return A2, cache
def computingcost(A2, Y, parameters):
    m = Y.shape[1]
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    logprobs = np.multiply(np.log(A2), Y) + np.multiply((1 - Y), np
    cost = - np.sum(logprobs) / m
    return cost
def backward_propagation(parameters, cache, X, Y):
    m = X.shape[1]
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    A1 = cache["A1"]
    A2 = cache["A2"]
    dZ2= A2 - Y
    cW2 = (1 / m) * np.dot(dZ2, A1.T)
    cb2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)
    dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.power(A1, 2))
    cW1 = (1 / m) * np.dot(dZ1, X.T)
    cW1 = (1 / m) * np.dot(dZ1, X.T)
    cb1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)
    grads = {"cW1": cW1,
    "cb1": cb1,
    "cW2": cW2,
    "cb2": cb2}
    return grads
def new_para(parameters, grads, lr=1.2):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
    cW1 = grads["cW1"]
    cb1 = grads["cb1"]
    cW2 = grads["cW2"]
    cb2 = grads["cb2"]
    W1 = W1 - lr * cW1
    b1 = b1 - lr * cb1
    W2 = W2 - lr * cW2
    b2 = b2 - lr * cb2
    parameters = {"W1": W1,
    "b1": b1,
    "W2": W2,
    "b2": b2}
    return parameters
def nn_model(X, Y, n_h, epoch=10000, print_cost=False):
    np.random.seed(3)
    n_x = size(X, Y)[0]
    n_y = size(X, Y)[2]
    parameters = old_para(n_x, n_h, n_y)
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]
for i in range(0, epoch):
    A2, cache = forward_propagation(X, parameters)
    cost = computingcost(A2, Y, parameters)
    grads = backward_propagation(parameters, cache, X, Y)
    parameters = new_para(parameters, grads)
if print_cost and i % 1000 == 0:
    print ("Cost after iteration %i: %f" % (i, cost))
    return parameters,n_h
parameters = nn_model(X,Y , n_h = 6, epoch=10000, print_cost=True)
def plot_decision_boundary(model, X, y):
    x_min, x_max = X[0, :].min() - 0.25, X[0, :].max() + 0.25
    y_min, y_max = X[1, :].min() - 0.25, X[1, :].max() + 0.25
    h = 0.01
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_mi
    Z = model(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
plt.ylabel("x2")
plt.xlabel("x1")
plt.scatter(X[0, :], X[1, :], c=y, cmap=plt.cm.Spectral)
plot_decision_boundary(lambda x: predict(parameters, x.T), X, Y[0, :
```

```
plot_decision_boundary(lambda x: predict(parameters, x.T), X, Y[0,:
plt.title("Boundary for hidden layer size " + str(6))
plt.xlabel("Petal Length")
plt.ylabel("Petal Width")
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/__init__.py in __getattr__(self, key)
    104         try:
--> 105             return self[key]
    106         except KeyError:

KeyError: 'loc'

During handling of the above exception, another exception occurred:

AttributeError                            Traceback (most recent call last)
<ipython-input-3-1a9fbeb35c56> in <module>
      4 from sklearn import datasets
      5 iris = datasets.load_iris()
----> 6 iris.loc[iris["Name"]=="virginica","species"]=0
      7 iris.loc[iris["Name"]=="versicolor","species"]=1
      8 iris.loc[iris["Name"]=="setosa","species"] = 2

~/opt/anaconda3/lib/python3.7/site-packages/sklearn/utils/__init__.py in __getattr__(self, key)
    105             return self[key]
    106         except KeyError:
--> 107             raise AttributeError(key)
    108 
    109     def __setstate__(self, state):

AttributeError: loc
```

In [ ]: