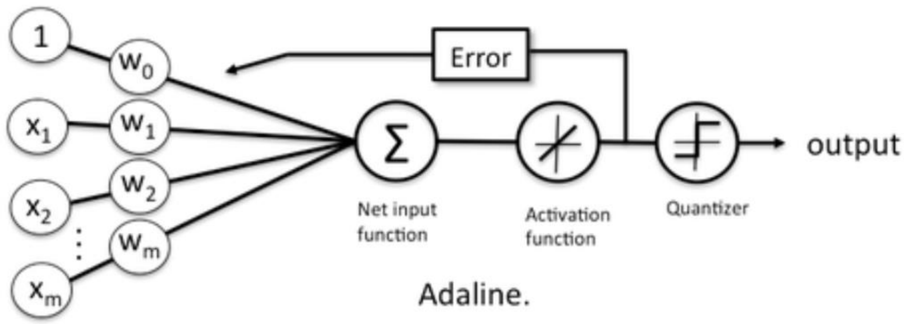
ADALINE

AND, OR logic gates and training on sample dataset

SRI SAI VIJAYA ADITYA NITTALA
177163
841775

The training of ADALINE follows the below diagram:



The algorithm is:

1. Initialize the weights and bias to a random number.
2. For each training iteration:
    a. Calculate the prediction Z as:
        i. $Z = W'X + b = \sum_{i=1}^{m} w_i x_i + b$
        ii. Where:
            1. Z = prediction
            2. W = weights
            3. X = Feature Vector
            4. b = bias
    b. Apply the activation function, which here is an identity function.
    c. Apply the Quantizer function, which is analogous to *threshold* in Perceptron learning algorithm, to get class labels for the training examples as O.
    d. Update the weights by performing backpropagation as:
        i. $W = W - (\frac{\alpha}{m})\Delta W$
        ii. $b = b - (\frac{\alpha}{m})(Z - Y)$
        iii. Where:
            1. $\Delta W = (Z - Y).X$
            2. Z = prediction
            3. Y = Correct labels
            4. X = Feature Vector
            5. α = learning rate
            6. m = no. of training examples.
3. Repeat for given number of iterations.

This algorithm is called ***Gradient Descent***, which finds the global minimum of the cost function J(W, b) with respect to the parameters W(weights) and b(bias):

$$J(W, b) = \frac{1}{m}\sum_{i=1}^{m}(z_i - y_i)^2$$

- The code below implements the above algorithm on a sample data set as well as AND, OR gate.
- The sample training dataset has been normalized as it helps with performance as well as improves accuracy. ***Z-score*** normalization was used:
    - $d_i = \left(\frac{x_i - \mu}{\sigma}\right)$
- Parameters chosen:
    - $\alpha = 0.1$
    - Number of iterations = 25

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mou
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
X = [0, 0, 0, 1, 1, 0, 1, 1]
X = np.reshape(X, (4, 2))
y_and = [-1, -1, -1, 1]
y_and = np.reshape(y_and, (4, 1))
y_or = [-1, 1, 1, 1]
y_or = np.reshape(y_or, (4, 1))

print(X)
print(X.shape)
```

```
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
(4, 2)
```

```
def threshold(Z):
    for i in range(Z.shape[0]):
        if Z[i][0] > 0:
            Z[i][0] = 1
        else:
            Z[i][0] = -1
    return Z
```

```
def normalization(X):
    mean = np.mean(X, axis=0)
    std = np.std(X, axis=0)
    X = (X - mean)/std
    return X
```

```
''' Back propagation implemetation '''

def backpropagation(w, b, X, Z, y, a):
    w = w - a*np.dot((Z - y).T, X)/Z.shape[0]
    b = b - a*np.sum(Z - y)/Z.shape[0]
    return w, b
```

```
'''Forward propagation implementation'''

def train(X, y, w, b, a, iters):
    for i in range(iters):
        Z = np.dot(X, w.T) + b
        Z = threshold(Z)
        w, b = backpropagation(w, b, X, Z, y, a)
    return w, b


def find_accuracy(X, Y, w, b):
    Z = np.dot(X, w.T) + b
    Z = threshold(Z)
    return accuracy_score(y_test, Z)


w = np.zeros(2)
w = np.expand_dims(w, axis=0)
b = -1
a = 0.1

'''AND gate implementation'''
print("AND gate implementation : ")
w, b = train(X, y_and, w, b, a, 6)
print("Bias : {}".format(b))
print("Weights : \n{}\n".format(w))
test = [1, 1]
ans = np.dot(test, w.T) + b
print(ans)

w = np.zeros(2)
w = np.expand_dims(w, axis=0)

''' OR gate implementation '''
print("OR gate implementation : ")
w, b = train(X, y_or, w, b, a, 6)
print("Bias : {}".format(b))
print("Weights : \n{}".format(w))
```

```
AND gate implementation :
Bias : -0.6999999999999997
Weights :
[[0.3 0.3]]

[-0.1]
OR gate implementation :
Bias : -0.1999999999999997
Weights :
[[0.3 0.3]]
```

```
''' Perceptron Training Algorithm with sample data '''
```

```
Data = pd.read_csv('/content/drive/My Drive/percep_data.csv')
Y = Data['Y']
X = Data[['X1', 'X2']]
X = normalization(X)
Y = np.expand_dims(Y, axis=1)
print("Shape of Y : {}".format(Y.shape))
print("Shape of X : {}".format(X.shape))
```

```
Shape of Y : (1000, 1)
Shape of X : (1000, 2)
```

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.1, random_state = 42)
```

```
w = np.random.randn(1, 2)
b = np.random.randn()
print("Shape of weights : {}".format(w.shape))
print("Shape of x_train : {}, y_train : {}".format(x_train.shape, y_train.shape))
print(x_train)
```

```
Shape of weights : (1, 2)
Shape of x_train : (900, 2), y_train : (900, 1)
           X1        X2
716  1.698851  0.625282
351 -1.409669  1.409476
936  1.057688 -1.133713
256  0.745459  1.545916
635  1.230740  0.650181
..        ...       ...
106  0.882481  0.818460
270 -1.442591  0.311410
860 -0.963468 -0.862202
435 -1.417628 -0.112863
102  0.697162  1.217073

[900 rows x 2 columns]
```

```
w, b = train(np.array(x_train), np.array(y_train), w, b, a, 25)
accuracy = find_accuracy(np.array(x_test), np.array(y_test), w, b)
print("Accuracy of model : {}".format(accuracy))
```

```
Accuracy of model : 0.98
```