Classification using Multilayer Perceptron to solve XOR problem

17BCE0136 R.S.Rahul Sai

In [1]:
```
import numpy as np

def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))
def sigmoid_prime(x):
    return sigmoid(x)*(1.0-sigmoid(x))
def tanh(x):
    return np.tanh(x)
def tanh_prime(x):
    return 1.0 - x**2
```

In [2]:
```
class MLP:

    def __init__(self, layers, activation='tanh'):
        if activation == 'sigmoid':
            self.activation = sigmoid
            self.activation_prime = sigmoid_prime
        elif activation == 'tanh':
            self.activation = tanh
            self.activation_prime = tanh_prime

        # Set weights
        self.weights = []

        for i in range(1, len(layers) - 1):
            r = 2*np.random.random((layers[i-1] + 1, layers[i] + 1)) -1
            self.weights.append(r)

        r = 2*np.random.random( (layers[i] + 1, layers[i+1])) - 1
        self.weights.append(r)

    def fit(self, X, y, learning_rate=0.2, epochs=100000):
        ones = np.atleast_2d(np.ones(X.shape[0]))
        X = np.concatenate((ones.T, X), axis=1)

        for k in range(epochs):
            if k % 10000 == 0: print('epochs:', k)

            i = np.random.randint(X.shape[0])
            a = [X[i]]

            for l in range(len(self.weights)):
                dot_value = np.dot(a[l], self.weights[l])
                activation = self.activation(dot_value)
                a.append(activation)

            error = y[i] - a[-1]
            deltas = [error * self.activation_prime(a[-1])]

            for l in range(len(a) - 2, 0, -1):
                deltas.append(deltas[-1].dot(self.weights[l].T)*self.activation_prime(a[l]))

            deltas.reverse()

            for i in range(len(self.weights)):
                layer = np.atleast_2d(a[i])
                delta = np.atleast_2d(deltas[i])
                self.weights[i] += learning_rate * layer.T.dot(delta)

    def predict(self, x):
        a = np.hstack((np.ones(1).T,np.array(x)))
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return np.asscalar(a)
```

In [3]:
```python
nn = MLP([2,2,1])

## defining the data
X = np.array([[0, 0],[0, 1],[1, 0],[1, 1]])
Y = np.array([0, 1, 1, 0])

## Fitting the MLP
nn.fit(X, Y)
```

```
epochs: 0
epochs: 10000
epochs: 20000
epochs: 30000
epochs: 40000
epochs: 50000
epochs: 60000
epochs: 70000
epochs: 80000
epochs: 90000
```

In [4]:
```python
for e in X:
    print("X= %d Y=%d Prediction=%f" %(e[0],e[1],nn.predict(e)))
```

```
X= 0 Y=0 Prediction=0.000044
X= 0 Y=1 Prediction=0.996016
X= 1 Y=0 Prediction=0.997521
X= 1 Y=1 Prediction=0.000021
```