Sheet 8_XOR_Neural_Network

May 17, 2019

1 Using Backpropagation algorithm to train a two layer XOR problem.

```
In [36]: #imports cell
         import numpy as np
         import matplotlib.pyplot as plt
         import sys
```

1.0.1 For the XOR problem introduced below, 4 data points exist in the dataset -

x 1 = (1, 0), x 2 = (1, 1), x 3 = (0, 1), x 4 = (0, 0).)

```
In [2]: # The training data.
        X = np.array([[1, 0],[1, 1],[0, 1],[0, 0]])

        # The labels for the training data.
        Y = np.array([[1],[0],[1],[0]])
```

Let us define a one hidden layer network with two input units, N hidden layer units and one output unit, with training set of D data samples.

1.0.2 Activation Function used here is Sigmoid. I tried Relu but it kept getting me bugs i think i implemented it wrongly.

```
In [135]: def sigmoid(f):
              return 1 / (1 + np.exp(-2 * f))


          def sigmoid_derivative(f):
              return f * (1 - f)
```

1.0.3 Forward Calculations

```
In [136]: def forward(input_layer, output_layer, hidden_weights, output_weights, bias):
              z2 = np.dot(input_layer, hidden_weights)
              a2 = sigmoid(z2)
              a2 = np.vstack((a2.T, bias)).T
              z3 = np.dot(a2, output_weights)
              a3 = sigmoid(z3)
              return a2, a3, hidden_weights, output_weights
```

1.1 Back Probagation: Gradient Descent

```
In [141]: def backprobagation(input_layer, output_layer, hidden_weights, output_weights, bias,
              for _ in range(max_iter):
                  a2, a3, hidden_weights, output_weights = forward(input_layer, output_layer, h
                  loss_a3 = output_layer - a3
                  loss_a2 = np.dot(loss_a3, output_weights[0:2, :].T) * sigmoid(np.dot(input_la
                  delta_a3 = loss_a3 * sigmoid_derivative(a3)
                  delta_a2 = loss_a2 * sigmoid_derivative(a2[:, 0:2])
                  # Updating weights
                  output_weights += np.dot(a2.T, delta_a3)
                  hidden_weights += np.dot(input_layer.T, delta_a2)
              return a3
```

```
In [143]: # Randomly initialising weights
          np.random.seed(1)
          hidden_weights = np.random.random((2, 2))
          output_weights = np.random.random((3, 1))
          # Bias term initializiation
          bias = np.ones((1, 4))
          # Number of iterations
          iterations = 33000
          print(backprobagation(X, Y, hidden_weights, output_weights, bias, iterations))
```

```
[[0.97508655]
 [0.05622335]
 [0.97508655]
 [0.00115668]]
```