# MatheMagic

# Backpropagation

***Step 0:*** Read input and output

```
import numpy as np
x = np.array([[1,0,1,0],[1,0,1,1],[0,1,0,1]])
y = np.array([1,1,0]).reshape(3,-1)
```

| x | | | |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |

| y |
|---|
| 1 |
| 1 |
| 0 |

***Step 1:*** Initialize weights and biases with random values

```
wh = np.random.rand(4,3)
bh = np.random.rand(3)
```

```
wout = np.random.rand(3,1)
bout = np.random.rand(1)
```

| **wh** | | |
|---|---|---|
| 0.51010119 | 0.08058941 | 0.81539472 |
| 0.94568682 | 0.24722592 | 0.1687749 |
| 0.20810438 | 0.88061006 | 0.01211583 |
| 0.19951766 | 0.16610815 | 0.15118213 |

| **bh** | | |
|---|---|---|
| 0.05253469 | 0.23126176 | 0.64622242 |

| **wout** |
|---|
| 0.48488218 |
| 0.65993833 |
| 0.84282406 |

| **bout** |
|---|
| 0.83612087 |

***Step 2:*** Calculate hidden layer input:

```
hidden_layer_input = x.dot(wh) + bh
```

| hidden_layer_input | | |
|---|---|---|
| 0.77074025 | 1.19246122 | 1.47373298 |
| 0.97025791 | 1.35856938 | 1.6249151 |
| 1.19773916 | 0.64459583 | 0.96617946 |

***Step 3:*** Sigmoid activation on hidden layer

```
def sigmoid(x):
    return (1/(1+np.exp(-x)))
hidden_layer_activation = sigmoid(hidden_layer_input)
```

| hidden_layer_activation | | |
|---|---|---|
| 0.683681 | 0.76718096 | 0.81362412 |
| 0.7251709 | 0.79552709 | 0.835471877 |
| 0.76812235 | 0.65579161 | 0.72435733 |

***Step 4:*** Perform linear and non-linear transformation of hidden layer activation at output layer

```
output_layer = hidden_layer_activation.dot(wout) + bout
output = sigmoid(output_layer)
```

| output |
|---|
| |

| |
|---|
| 0.91369898 |
| 0.91810681 |
| 0.904810649 |

***Step 5:*** Calculate Error E

```
E = y - output
```

| **E** |
|---|
| 0.08630102 |
| 0.0818931 |
| -0.90481064 |

***Step 6:*** Compute slope at output and hidden layer

```
def derivative_sigmoid(x):
    sigmoid_X = sigmoid(x)
    return (sigmoid_X*(1-sigmoid_X))
slope_output_layer = derivative_sigmoid(output)
slope_hidden_layer = derivative_sigmoid(hidden_layer_activ
ation)
```

| **slope_output_layer** |
|---|
| 0.20430816 |
| 0.20392272 |

| |
|---|
| 0.20508267 |

| **slope_hidden_layer** | | |
|---|---|---|
| 0.22292008 | 0.21654365 | 0.21279683 |
| 0.21981279 | 0.21427262 | 0.21098953 |
| 0.21646905 | 0.22493655 | 0.21987491 |

***Step 7:*** Compute delta at output layer

```
lr = 1
d_output = E * slope_output_layer * lr
```

| **d_output** |
|---|
| 0.017632 |
| 0.01669988 |
| -0.18556098 |

***Step 8:*** Calculate Error at hidden layer

```
error_at_hidden_layer = d_output * np.transpose(wout)
```

| **error_at_hidden_layer** | | |
|---|---|---|
| 0.00854944 | 0.01163604 | 0.01486068 |
| | | |

| | | |
|---|---|---|
| 0.00809747 | 0.01102089 | 0.01407506 |
| -0.08997521 | -0.12245881 | -0.15639526 |

***Step 9:*** Compute delta at hidden layer

```
d_hidden_layer = error_at_hidden_layer * slope_hidden_laye
r
```

| **d_hidden_layer** | | |
|---|---|---|
| 0.00190584 | 0.00251971 | 0.0031623 |
| 0.00177993 | 0.00236148 | 0.00296969 |
| -0.01947685 | -0.02754546 | -0.03438739 |

***Step 10:*** Update weight at both output and hidden layer

```
#Updating weights
wout =  wout + (np.transpose(hidden_layer_activation).dot(
d_output))*lr
wh = wh + (np.transpose(x).dot(d_hidden_layer))*lr
```

| **wh** | | |
|---|---|---|
| 0.51378696 | 0.08547059 | 0.82152672 |
| 0.92620997 | 0.21968046 | 0.13438751 |
| 0.21179015 | 0.88549124 | 0.01824783 |
| | | |

| | | |
|---|---|---|
| 0.18182074 | 0.14092417 | 0.11976443 |

| **wout** |
|---|
| 0.12977637 |
| 0.37530676], |
| 0.524481 |

***Step 11:*** Update biases at both output and hidden layer

```
#Updating bias
bh = bh + np.sum(d_hidden_layer,axis=0)*lr
bout = bout + np.sum(d_output,axis=0)*lr
```

| **bout** |
|---|
| 0.68489177 |

| **bh** | | |
|---|---|---|
| 0.03674361 | 0.20859748 | 0.61796702 |