assignment1

February 27, 2020

[2]:
```
#=============================================================================================
# Assignment No.1: Back Propagation algorithm
# Name: Mahesh Tatyasaheb Chavan
# Roll No. BETB65
#=============================================================================================

import numpy as np

def relu(z):
    a=np.maximum(0,z[0])
    b=np.maximum(0,z[1])
    c=np.maximum(0,z[2])
    d=np.maximum(0,z[3])
    return([a,b,c,d])

def relu_derivative(z):
    y=[]
    for i in range(0,len(z)):
        if(z[i]==0 or z[i]<0):
            y.append(0)
        else:
            y.append(z[i])
    return y
```

[3]:
```
training_inputs = np.array([[0,0,1],
                            [1,1,1],
                            [1,0,1],
                            [0,1,1]])
training_outputs = np.array([[0,1,1,0]]).T
#np.random.seed(1)
#synaptic_weights=2*np.random.random((3,1))-1

synaptic_weights = [[0.1],[0.5],[0.2]]
print(training_inputs)
print(training_outputs)
print(synaptic_weights)
```

```
#s=[-4,-3,-2,-1,0,1,2,3,4]
#relu_derivative(s)
```

```
[[0 0 1]
 [1 1 1]
 [1 0 1]
 [0 1 1]]
[[0]
 [1]
 [1]
 [0]]
[[0.1], [0.5], [0.2]]
```

[4]:
```
input_layer = training_inputs
outputs=relu(np.dot(input_layer,synaptic_weights))
print('outputs\n',np.array(outputs))
y=relu_derivative(outputs)
z=[]
for i in y:
    z.append(i)
print('z',np.array(z))
error=training_outputs-outputs
print('error\n',error)
adjustments=error*z
print('adjustments\n',adjustments)
synaptic_weights=synaptic_weights + np.dot(input_layer.T,adjustments)
print('synaptics',synaptic_weights)
```

```
outputs
 [[0.2]
 [0.8]
 [0.3]
 [0.7]]
z [[0.2]
 [0.8]
 [0.3]
 [0.7]]
error
 [[-0.2]
 [ 0.2]
 [ 0.7]
 [-0.7]]
adjustments
 [[-0.04]
 [ 0.16]
 [ 0.21]
 [-0.49]]
```

```
synaptics [[0.47]
 [0.17]
 [0.04]]
```

[5]:
```
for j in range(3):
    input_layer = training_inputs
    outputs=relu(np.dot(input_layer,synaptic_weights))
    print('output\n',np.array(outputs))
    print('outputs\n',np.array(outputs))
    y=relu_derivative(outputs)
    z=[]
    for i in y:
        z.append(i)
    error=training_outputs-outputs
    print('error\n',error)
    adjustments=error
    print('adjustments\n',adjustments)
    synaptic_weights=synaptic_weights + np.dot(input_layer.T,adjustments)
    print('updated synaptics_weights\n',synaptic_weights)

print('updated synaptics_weights\n',synaptic_weights)
print('outputs\n',np.array(outputs))
```

```
output
 [[0.04]
 [0.68]
 [0.51]
 [0.21]]
outputs
 [[0.04]
 [0.68]
 [0.51]
 [0.21]]
error
 [[-0.04]
 [ 0.32]
 [ 0.49]
 [-0.21]]
adjustments
 [[-0.04]
 [ 0.32]
 [ 0.49]
 [-0.21]]
updated synaptics_weights
 [[1.28]
 [0.28]
 [0.6 ]]
output
```

```
 [[0.6 ]
 [2.16]
 [1.88]
 [0.88]]
outputs
 [[0.6 ]
 [2.16]
 [1.88]
 [0.88]]
error
 [[-0.6 ]
 [-1.16]
 [-0.88]
 [-0.88]]
adjustments
 [[-0.6 ]
 [-1.16]
 [-0.88]
 [-0.88]]
updated synaptics_weights
 [[-0.76]
 [-1.76]
 [-2.92]]
output
 [[0.]
 [0.]
 [0.]
 [0.]]
outputs
 [[0.]
 [0.]
 [0.]
 [0.]]
error
 [[0.]
 [1.]
 [1.]
 [0.]]
adjustments
 [[0.]
 [1.]
 [1.]
 [0.]]
updated synaptics_weights
 [[ 1.24]
 [-0.76]
 [-0.92]]
updated synaptics_weights
```

```
 [[ 1.24]
 [-0.76]
 [-0.92]]
outputs
 [[0.]
 [0.]
 [0.]
 [0.]]
```

[ ]:

1 **In the ReLU Activation function whenever the updated weights will be negative the output will be zero**

2 **To overcome this problem we are using the leaky relu activation function**

[ ]: