BooleanFunction

January 30, 2019

```
In [1]: import numpy as np
        from numpy.random import rand as U

In [2]: x_train = np.array([[0, 0, 0],
                            [0, 0, 1],
                            [0, 1, 0],
                            [0, 1, 1],
                            [1, 0, 0],
                            [1, 0, 1],
                            [1, 1, 0],
                            [1, 1, 1],])

        y_train = np.zeros((8,1))

In [3]: def activation(x):
            return 1/(1+np.exp(-x))

In [4]: def Delta(x, y, o):
            dW = x*y*(y - o)*(y - 1)
            return (np.sum(dW, axis = 0)).reshape((x.shape[1],1))

In [5]: def output(x, y, z):
            y = (activation(np.dot(np.array([[x, y, z]]), W) + b))
            y = y > 0.5
            return int(y)
```

Please provide your 3 input binary function.
==========================| **NOTE** |========================== - Use x, y, z as 3 inputs and for writing not of x write it as x' - use '.' for 'AND' and '+' for 'OR' operators. - write the function as sum of products.
===
Example: x'.y.z + y'.z + z'

```
In [28]: print("- Use x, y, z as 3 inputs and for writing not of x write it as x' \n- use '.' f

         f = input("Function: ")
         bool_exp = ""
         # convert the inputed function into string mathematical expression
         # The expression gives > 0 for logic 1 and 0 for logic 0
         # eval is used to convert string into expression!
         for j in range(len(f)):
             if f[j] == "'":
                 bool_exp = bool_exp[:-1] + "(1 - " + str(bool_exp[-1]) + ")"
             elif f[j] == ".":
                 bool_exp += "*"
             else:
                 bool_exp += f[j]

         print(bool_exp)
```

```
- Use x, y, z as 3 inputs and for writing not of x write it as x'
- use '.' for 'AND' and '+' for 'OR' operators.
- write the function as sum of products.
Example: x'.y.z + y'.z + z'
Function: x'.y'.z'
(1 - x)*(1 - y)*(1 - z)
```

```
In [29]: for i in range(8):
             x = x_train[i][0]
             y = x_train[i][1]
             z = x_train[i][2]

             y_train[i] = not (not (eval(bool_exp)))
```

```
In [30]: W = U(3,1) - 0.5
         b = U(1,1) - 0.5
         lr = 1

         print("=======[ Start Learning ]=======")
         for e in range(1000):
             y = (activation(np.dot(x_train, W) + b)).reshape(-1,1)
             if e%100 == 0:
                 error = np.sum((y-y_train)*(y-y_train))/8
                 print('Epoch: ', e, 'Error: ', error)
             dW = Delta(x_train, y, y_train)
             db = Delta(np.ones((8,1)), y, y_train)

             W += lr*dW
             b += lr*db
```

```
=======[ Start Learning ]=======
Epoch:  0 Error:  0.3116370118727666
Epoch:  100 Error:  0.014138568640880428
Epoch:  200 Error:  0.006330201325301979
Epoch:  300 Error:  0.003932848717290524
```

```
Epoch:  400 Error:  0.0028151897758026284
Epoch:  500 Error:  0.002178296642620836
Epoch:  600 Error:  0.0017700285600263088
Epoch:  700 Error:  0.0014872862394903477
Epoch:  800 Error:  0.0012804881470882138
Epoch:  900 Error:  0.0011229699240297903
```

```
In [31]: print('Veirfy f = ' + f + '\n')
         print('[x y z] [ f ]')
         for test in x_train:
             print(test, '[', output(test[0], test[1], test[2]), ']')
```

```
Veirfy f = x'.y'.z'

[x y z] [ f ]
[0 0 0] [ 1 ]
[0 0 1] [ 0 ]
[0 1 0] [ 0 ]
[0 1 1] [ 0 ]
[1 0 0] [ 0 ]
[1 0 1] [ 0 ]
[1 1 0] [ 0 ]
[1 1 1] [ 0 ]
```

```
In [ ]:
```