# XOR Gate

```
import numpy as np
```

```
x = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])
```

```
print(x)
```

```
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
```

```
print(y)
```

```
[[0]
 [1]
 [1]
 [0]]
```

```
np.random.seed(10)
weights = np.random.random((2,1))
weights
```

```
array([[0.77132064],
       [0.02075195]])
```

```
sum = np.dot(x,weights) - 1
sum
```

```
array([[-1.        ],
       [-0.97924805],
       [-0.22867936],
       [-0.20792741]])
```

```
def activation(x):
  return 1/(1+np.exp(-x))
```

```
ypred = activation(sum)
ypred
```

```
array([[0.26894142],
       [0.27304101],
       [0.443078  ],
       [0.44820462]])
```

```
ypred.round()
```

```
array([[0.],
       [0.],
       [0.],
       [0.]])
```

```
error = y - ypred
error
```

```
array([[-0.26894142],
       [ 0.72695899],
       [ 0.556922  ],
       [-0.44820462]])
```

```
#applying gradient descent to minimize the prediction error
def gradient(a,x):
  return a * x * (1-x)
```

```
chng_in_weight = gradient(error,ypred)      ## Apply Gradient Descent
weights = weights + np.dot(x.T,chng_in_weight)  ## Weight updation
weights
```

```
array([[0.79789792],
       [0.05419703]])
```

```
for i in range(2000):
  sum = np.dot(x,weights) - 1
  ypred = activation(sum)
  error = y - ypred
  chng_in_weight = gradient(error,ypred)      ## Apply Gradient Descent
  weights = weights + np.dot(x.T,chng_in_weight)
  if i%50 == 0:
    print(error,end="\n\n")
```

```
[[-0.26894142]
 [ 0.72027034]
 [ 0.55035424]
 [-0.463091  ]]

[[-0.26894142]
 [ 0.60413742]
 [ 0.56420998]
 [-0.5790804 ]]

[[-0.26894142]
 [ 0.58589864]
 [ 0.57934362]
 [-0.58246106]]

[[-0.26894142]
 [ 0.58310292]
 [ 0.58204024]
 [-0.58256726]]
```

```
      [[-0.26894142]
       [ 0.58265635]
       [ 0.58248414]
       [-0.58257013]]

      [[-0.26894142]
       [ 0.58258416]
       [ 0.58255625]
       [-0.5825702 ]]

      [[-0.26894142]
       [ 0.58257247]
       [ 0.58256795]
       [-0.58257021]]

      [[-0.26894142]
       [ 0.58257057]
       [ 0.58256984]
       [-0.58257021]]

      [[-0.26894142]
       [ 0.58257027]
       [ 0.58257015]
       [-0.58257021]]

      [[-0.26894142]
       [ 0.58257022]
       [ 0.5825702 ]
       [-0.58257021]]

      [[-0.26894142]
       [ 0.58257021]
       [ 0.5825702 ]
       [-0.58257021]]

      [[-0.26894142]
       [ 0.58257021]
       [ 0.58257021]
```

```
ypred
```

```
     array([[0.26894142],
            [0.41742979],
            [0.41742979],
            [0.58257021]])
```

```
ypred.round()
```

```
     array([[0.],
            [0.],
            [0.],
            [1.]])
```

```
prediction_error=(y-ypred)
prediction_error
```

```
array([[-0.26894142],
       [ 0.58257021],
       [ 0.58257021],
       [-0.58257021]])
```

```
weights
```

```
array([[0.66666667],
       [0.66666667]])
```

## NAND

```
import numpy as np
```

```
x = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[1],[1],[1],[0]])
```

```
print(x)
```

```
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
```

```
print(y)
```

```
[[1]
 [1]
 [1]
 [0]]
```

```
np.random.seed(10)
weights = np.random.random((2,1))
weights
```

```
array([[0.77132064],
       [0.02075195]])
```

```
sum = np.dot(x,weights) -3
sum
```

```
array([[-3.        ],
       [-2.97924805],
       [-2.22867936],
       [-2.20792741]])
```

```
def activation(x):
  return 1/(1+np.exp(-x))
```

```
ybar = activation(sum)
ybar
```

```
array([[0.04742587],
       [0.04837223],
       [0.09720447],
       [0.09904086]])
```

```
ybar.round()
```

```
array([[0.],
       [0.],
       [0.],
       [0.]])
```

```
error = y - ybar
error
```

```
array([[ 0.95257413],
       [ 0.95162777],
       [ 0.90279553],
       [-0.09904086]])
```

```
def gradient(a,x):
  return a * x * (1-x)
```

```
change_in_weight = gradient(error,ybar)
weights = weights + np.dot(x.T,change_in_weight)
weights
```

```
array([[0.84170856],
       [0.05572003]])
```

```
for i in range(1000):
  sum = np.dot(x,weights) -3
  ybar = activation(sum)
  error = y - ybar
  change_in_weight = gradient(error,ybar)
  weights = weights + np.dot(x.T,change_in_weight)
  if i%50 == 0:
    print(error,end="\n\n")
```

```
[[ 0.95257413]
 [ 0.94999245]
 [ 0.89644104]
 [-0.10884715]]

[[ 0.95257413]
 [ 0.98845575]
 [ 0.36738421]
 [-0.28771636]]

[[ 0.95257413]
```

```
        [ 0.9975499 ]
        [ 0.20197546]
        [-0.16312178]]

       [[ 0.95257413]
        [ 0.99884105]
        [ 0.14674279]
        [-0.11933907]]

       [[ 0.95257413]
        [ 0.99927604]
        [ 0.11910708]
        [-0.09716383]]

       [[ 0.95257413]
        [ 0.99948388]
        [ 0.10219146]
        [-0.08351316]]

       [[ 0.95257413]
        [ 0.99960305]
        [ 0.09059917]
        [-0.07412676]]

       [[ 0.95257413]
        [ 0.99967943]
        [ 0.08206627]
        [-0.06720191]]

       [[ 0.95257413]
        [ 0.99973218]
        [ 0.07546921]
        [-0.06183927]]

       [[ 0.95257413]
        [ 0.99977063]
        [ 0.07018322]
        [-0.05753697]]

       [[ 0.95257413]
        [ 0.9997998 ]
        [ 0.06583122]
        [-0.0539913 ]]

       [[ 0.95257413]
        [ 0.99982263]
        [ 0.06217106]
```

```
ybar
```

```
     array([[4.74258732e-02],
            [8.59419912e-05],
            [9.56003557e-01],
            [3.61555583e-02]])
```

```
ybar.round()
```

```
array([[0.],
       [0.],
       [1.],
       [0.]])
```

```
weights
```

```
array([[ 6.0792434],
       [-6.3629261]])
```

```
prediction_error=(y-ybar)
prediction_error
```

```
array([[ 0.95257413],
       [ 0.99991406],
       [ 0.04399644],
       [-0.03615556]])
```

## XNOR

```
import numpy as np
```

```
x = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[1],[0],[0],[1]])
```

```
print(x)
```

```
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
```

```
print(y)
```

```
[[1]
 [0]
 [0]
 [1]]
```

```
np.random.seed(10)
weights = np.random.random((2,1))
weights
```

```
array([[0.77132064],
       [0.02075195]])
```

```
sum = np.dot(x,weights) -3
sum
```

```
array([[-3.        ],
       [-2.97924805],
       [-2.22867936],
       [-2.20792741]])
```

```
def activation(x):
  return 1/(1+np.exp(-x))
```

```
ybar = activation(sum)
ybar
```

```
array([[0.04742587],
       [0.04837223],
       [0.09720447],
       [0.09904086]])
```

```
ybar.round()
```

```
array([[0.],
       [0.],
       [0.],
       [0.]])
```

```
error = y - ybar
error
```

```
array([[ 0.95257413],
       [-0.04837223],
       [-0.09720447],
       [ 0.90095914]])
```

```
def gradient(a,x):
  return a * x * (1-x)
```

```
change_in_weight = gradient(error,ybar)
weights = weights + np.dot(x.T,change_in_weight)
weights
```

```
array([[0.84318457],
       [0.09891944]])
```

```
for i in range(1000):
  sum = np.dot(x,weights) -3
  ybar = activation(sum)
  error = y - ybar
  change_in_weight = gradient(error,ybar)
  weights = weights + np.dot(x.T,change_in_weight)
  if i%50 == 0:
    print(error,end="\n\n")
```

```
[[ 0.95257413]
 [-0.05210017]
 [-0.10369606]
 [ 0.88674304]]

[[ 0.95257413]
 [-0.26405347]
 [-0.27377676]
 [ 0.26904963]]

[[ 0.95257413]
 [-0.26875444]
 [-0.26912837]
 [ 0.26894154]]

[[ 0.95257413]
 [-0.26893423]
 [-0.26894861]
 [ 0.26894142]]

[[ 0.95257413]
 [-0.26894115]
 [-0.2689417 ]
 [ 0.26894142]]

[[ 0.95257413]
 [-0.26894141]
 [-0.26894143]
 [ 0.26894142]]

[[ 0.95257413]
 [-0.26894142]
 [-0.26894142]
 [ 0.26894142]]

[[ 0.95257413]
 [-0.26894142]
 [-0.26894142]
 [ 0.26894142]]

[[ 0.95257413]
 [-0.26894142]
 [-0.26894142]
 [ 0.26894142]]

[[ 0.95257413]
 [-0.26894142]
 [-0.26894142]
 [ 0.26894142]]

[[ 0.95257413]
 [-0.26894142]
 [-0.26894142]
 [ 0.26894142]]

[[ 0.95257413]
 [-0.26894142]
 [-0.26894142]
```

```
ybar
```

```
array([[0.04742587],
       [0.26894142],
       [0.26894142],
       [0.73105858]])
```

```
ybar.round()
```

```
array([[0.],
       [0.],
       [0.],
       [1.]])
```

```
error = y - ybar
error
```

```
array([[ 0.95257413],
       [-0.26894142],
       [-0.26894142],
       [ 0.26894142]])
```

```
weights
```

```
array([[2.],
       [2.]])
```

```
prediction_error=(y-ybar)
prediction_error
```

```
array([[ 0.95257413],
       [-0.26894142],
       [-0.26894142],
       [ 0.26894142]])
```

NOR

```
import numpy as np
```

```
x = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[1],[0],[0],[0]])
```

```
print(x)
print(y)
```

```
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
[[1]
 [0]
```

```
       [0]
       [0]]
```

```
np.random.seed(10)
weights = np.random.random((2,1))
weights
```

```
array([[0.77132064],
       [0.02075195]])
```

```
sum = np.dot(x,weights) - 1
sum
```

```
array([[-1.        ],
       [-0.97924805],
       [-0.22867936],
       [-0.20792741]])
```

```
def activation(x):
  return 1/(1+np.exp(-x))
```

```
ypred = activation(sum)
ypred
```

```
array([[0.26894142],
       [0.27304101],
       [0.443078  ],
       [0.44820462]])
```

```
ypred.round()
```

```
array([[0.],
       [0.],
       [0.],
       [0.]])
```

```
error = y - ybar
error
```

```
array([[ 0.95257413],
       [-0.26894142],
       [-0.26894142],
       [-0.73105858]])
```

```
def gradient(a,x):
  return a * x * (1-x)
```

```
change_in_weight = gradient(error,ybar)
weights = weights + np.dot(x.T,change_in_weight)
weights
```

```
array([[ 0.57470871],
       [-0.17585998]])
```

```
for i in range(1000):
  sum = np.dot(x,weights) -3
  ybar = activation(sum)
  error = y - ybar
  change_in_weight = gradient(error,ybar)
  weights = weights + np.dot(x.T,change_in_weight)
  if i%50 == 0:
    print(error,end="\n\n")
```

```
[[ 0.95257413]
 [-0.04008433]
 [-0.08126433]
 [-0.06906436]]

[[ 0.95257413]
 [-0.0331819 ]
 [-0.05861417]
 [-0.04115506]]

[[ 0.95257413]
 [-0.02986544]
 [-0.04871087]
 [-0.03068997]]

[[ 0.95257413]
 [-0.0277031 ]
 [-0.0427598 ]
 [-0.02492671]]

[[ 0.95257413]
 [-0.02610577]
 [-0.03865761]
 [-0.02119156]]

[[ 0.95257413]
 [-0.02484266]
 [-0.03560039]
 [-0.01853863]]

[[ 0.95257413]
 [-0.02380017]
 [-0.03320357]
 [-0.01653984]]

[[ 0.95257413]
 [-0.0229142 ]
 [-0.03125635]
 [-0.01497042]]

[[ 0.95257413]
 [-0.02214503]
 [-0.02963196]
 [-0.01369993]]
```

```
       [[ 0.95257413]
        [-0.02146639]
        [-0.0282489 ]
        [-0.01264695]]

       [[ 0.95257413]
        [-0.02085996]
        [-0.02705199]
        [-0.01175777]]

       [[ 0.95257413]
        [-0.02031248]
        [-0.02600234]
```

```
ybar
```

```
       array([[0.04742587],
              [0.01694162],
              [0.02014032],
              [0.00706452]])
```

```
ybar.round()
```

```
       array([[0.],
              [0.],
              [0.],
              [0.]])
```

```
error = y - ybar
error
```

```
       array([[ 0.95257413],
              [-0.01694162],
              [-0.02014032],
              [-0.00706452]])
```

```
weights
```

```
       array([[-0.88513239],
              [-1.06122664]])
```

```
prediction_error=(y-ybar)
prediction_error
```

```
       array([[ 0.95257413],
              [-0.01694162],
              [-0.02014032],
              [-0.00706452]])
```

## AND Gate

```
import numpy as np
```

```
x = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[0],[0],[1]])


print(x)
```

```
     [[0 0]
      [0 1]
      [1 0]
      [1 1]]
```

```
print(y)
```

```
     [[0]
      [0]
      [0]
      [1]]
```

```
np.random.seed(10)
weights = np.random.random((2,1))
weights
```

```
     array([[0.77132064],
            [0.02075195]])
```

```
sum = np.dot(x,weights) -3
sum
```

```
     array([[-3.        ],
            [-2.97924805],
            [-2.22867936],
            [-2.20792741]])
```

```
def activation(x):
  return 1/(1+np.exp(-x))


ybar = activation(sum)
ybar
```

```
     array([[0.04742587],
            [0.04837223],
            [0.09720447],
            [0.09904086]])
```

```
ybar.round()
```

```
     array([[0.],
            [0.],
            [0.],
            [0.]])
```

```
error = y - ybar
error
```

```
array([[-0.04742587],
       [-0.04837223],
       [-0.09720447],
       [ 0.90095914]])
```

```
def gradient(a,x):
  return a * x * (1-x)
```

```
change_in_weight = gradient(error,ybar)
weights = weights + np.dot(x.T,change_in_weight)
weights
```

```
array([[0.84318457],
       [0.09891944]])
```

```
for i in range(1000):
  sum = np.dot(x,weights) -3
  ybar = activation(sum)
  error = y - ybar
  change_in_weight = gradient(error,ybar)
  weights = weights + np.dot(x.T,change_in_weight)
  if i%50 == 0:
    print(error,end="\n\n")
```

```
[[-0.04742587]
 [-0.05210017]
 [-0.10369606]
 [ 0.88674304]]

[[-0.04742587]
 [-0.26405347]
 [-0.27377676]
 [ 0.26904963]]

[[-0.04742587]
 [-0.26875444]
 [-0.26912837]
 [ 0.26894154]]

[[-0.04742587]
 [-0.26893423]
 [-0.26894861]
 [ 0.26894142]]

[[-0.04742587]
 [-0.26894115]
 [-0.2689417 ]
 [ 0.26894142]]

[[-0.04742587]
 [-0.26894141]
 [-0.26894143]
 [ 0.26894142]]
```

```
      [[-0.04742587]
       [-0.26894142]
       [-0.26894142]
       [ 0.26894142]]

      [[-0.04742587]
       [-0.26894142]
       [-0.26894142]
       [ 0.26894142]]

      [[-0.04742587]
       [-0.26894142]
       [-0.26894142]
       [ 0.26894142]]

      [[-0.04742587]
       [-0.26894142]
       [-0.26894142]
       [ 0.26894142]]

      [[-0.04742587]
       [-0.26894142]
       [-0.26894142]
       [ 0.26894142]]

      [[-0.04742587]
       [-0.26894142]
       [-0.26894142]
```

```
ybar
```

```
      array([[0.04742587],
             [0.26894142],
             [0.26894142],
             [0.73105858]])
```

```
ybar.round()
```

```
      array([[0.],
             [0.],
             [0.],
             [1.]])
```

```
weights
```

```
      array([[2.],
             [2.]])
```

```
prediction_error=(y-ybar)
prediction_error
```

```
      array([[-0.04742587],
             [-0.26894142],
             [-0.26894142],
             [ 0.26894142]])
```

## OR Gate

```
import numpy as np
```

```
x = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[1]])
print(x)
print(y)
```

```
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
[[0]
 [1]
 [1]
 [1]]
```

```
np.random.seed(10)   ## fix thr random value
weights = np.random.random((2,1))
weights
```

```
array([[0.77132064],
       [0.02075195]])
```

```
sum = np.dot(x,weights) - 1
sum
```

```
array([[-1.        ],
       [-0.97924805],
       [-0.22867936],
       [-0.20792741]])
```

```
def activation(x):
  return 1/(1+np.exp(-x))
```

```
ypred = activation(sum)
ypred
```

```
array([[0.26894142],
       [0.27304101],
       [0.443078  ],
       [0.44820462]])
```

```
ypred.round()
```

```
array([[0.],
       [0.],
       [0.],
       [0.]])
```

```
error = y - ypred
error
```

```
array([[-0.26894142],
       [ 0.72695899],
       [ 0.556922  ],
       [ 0.55179538]])
```

```
#applying gradient descent to minimize the prediction error
def gradient(a,x):
  return a * x * (1-x)
```

```
chng_in_weight = gradient(error,ypred)      ## Apply Gradient Descent
weights = weights + np.dot(x.T,chng_in_weight)  ## Weight updation
weights
```

```
array([[1.04521516],
       [0.30151427]])
```

```
for i in range(2000):
  sum = np.dot(x,weights) - 1
  ypred = activation(sum)
  error = y - ypred
  chng_in_weight = gradient(error,ypred)      ## Apply Gradient Descent
  weights = weights + np.dot(x.T,chng_in_weight)
  if i%50 == 0:
    print(error,end="\n\n")
```

```
[[-0.26894142]
 [ 0.66785195]
 [ 0.48869814]
 [ 0.41417575]]

[[-0.26894142]
 [ 0.1247141 ]
 [ 0.1159731 ]
 [ 0.00682947]]

[[-0.26894142]
 [ 0.08271837]
 [ 0.07978869]
 [ 0.00286821]]

[[-0.26894142]
 [ 0.06558036]
 [ 0.06404146]
 [ 0.0017635 ]]

[[-0.26894142]
 [ 0.05580141]
 [ 0.05482578]
 [ 0.00125954]]

[[-0.26894142]
```

```
       [ 0.04931368]
       [ 0.0486278 ]
       [ 0.00097442]]

      [[-0.26894142]
       [ 0.04462013]
       [ 0.04410539]
       [ 0.00079213]]

      [[-0.26894142]
       [ 0.04102766]
       [ 0.04062359]
       [ 0.000666  ]]

      [[-0.26894142]
       [ 0.03816673]
       [ 0.03783894]
       [ 0.00057376]]

      [[-0.26894142]
       [ 0.03582034]
       [ 0.03554769]
       [ 0.00050349]]

      [[-0.26894142]
       [ 0.03385175]
       [ 0.03362044]
       [ 0.00044823]]

      [[-0.26894142]
       [ 0.03217004]
       [ 0.03197066]
```

```
ypred
```

```
      array([[0.26894142],
             [0.9836511 ],
             [0.98367829],
             [0.99989856]])
```

```
ypred.round()
```

```
      array([[0.],
             [1.],
             [1.],
             [1.]])
```

```
prediction_error=(y-ypred)
prediction_error
```

```
      array([[-2.68941421e-01],
             [ 1.63489035e-02],
             [ 1.63217074e-02],
             [ 1.01442866e-04]])
```

```
weights
```

```
array([[5.099065  ],
       [5.09737335]])
```