EXPERIMENT 1

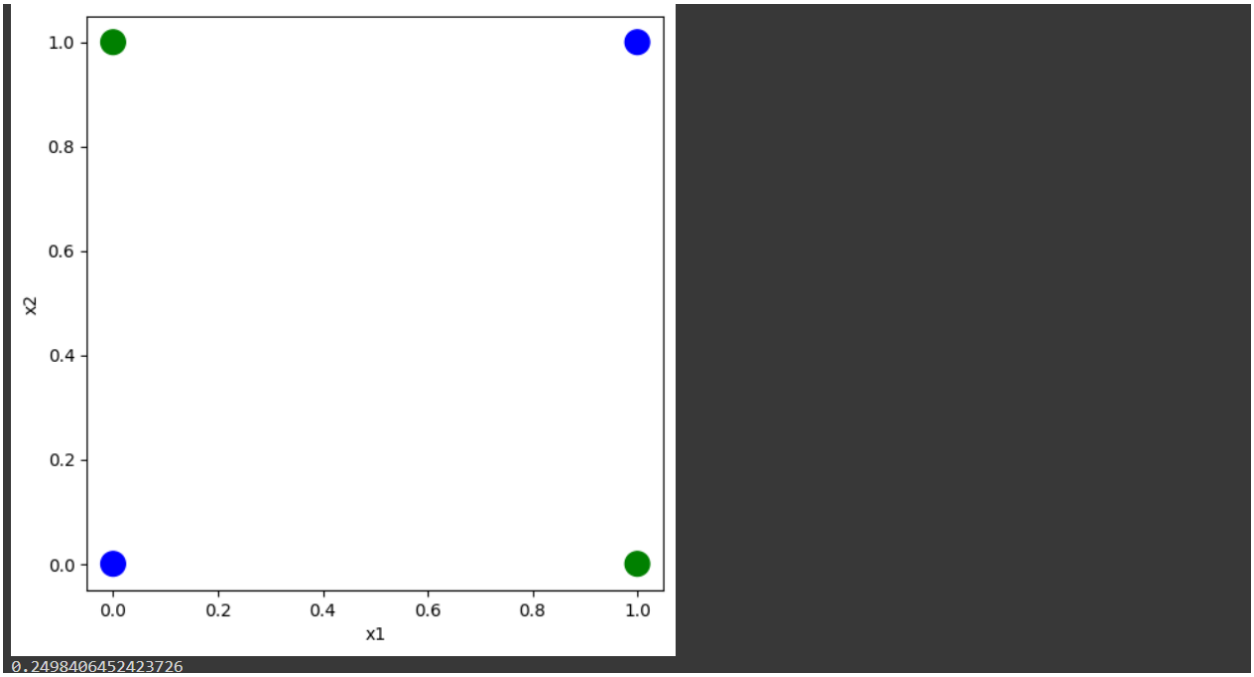
Build Your Own MLP – XOR

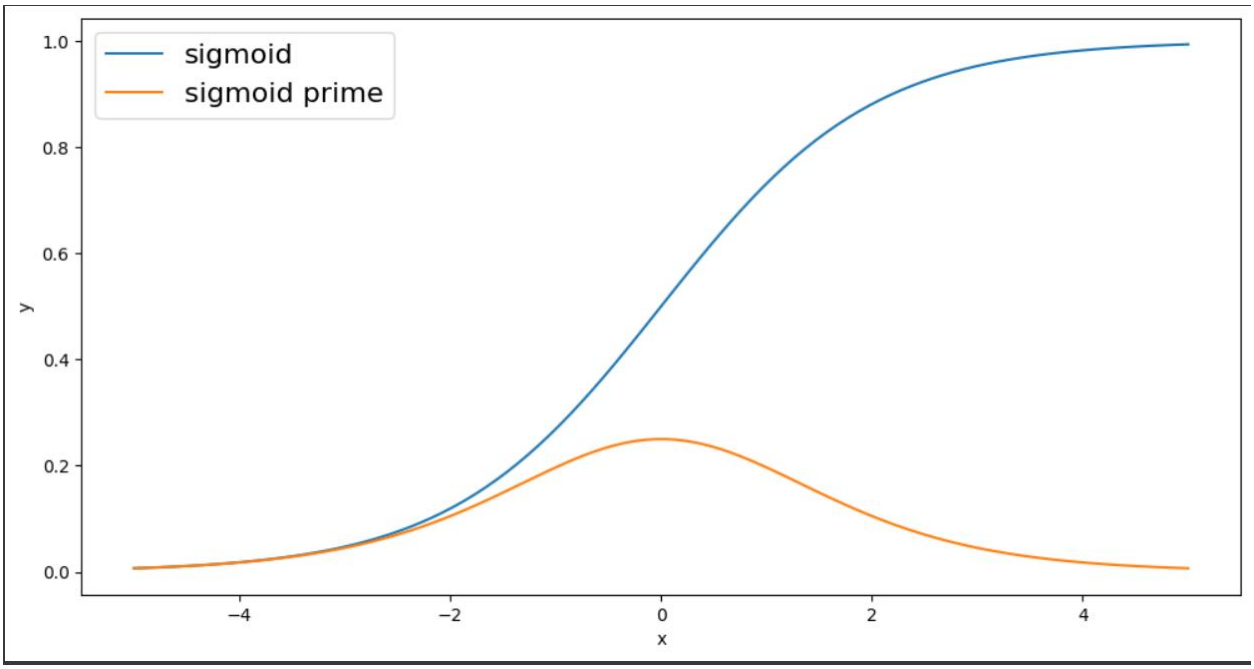
CODE:

```
# -*- coding: utf-8 -*-
"""XOR_exp1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/10qYQCeThQYV7qjoKDp70zk2utidVqF_8
"""

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import matplotlib.pyplot as plt
from pylab import rcParams
# %matplotlib inline
rcParams['figure.figsize'] = 12, 6
RANDOM_SEED = 56
np.random.seed(RANDOM_SEED)
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([ [0], [1], [1], [0]])
# Colors corresponding to class labels y.
colors = ['green' if y_ == 1 else 'blue' for y_ in y]
fig = plt.figure()
fig.set_figwidth(6)
fig.set_figheight(6)
plt.scatter(X[:,0],X[:,1],s=200,c=colors)
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
def sigmoid(x):
 """
 Computes the sigmoid function sigm(input) = 1/(1+exp(-input))
 """
 return 1 / (1 + np.exp(-x))
def sigmoid_(y):
 """
 Computes the derivative of sigmoid funtion. sigmoid(y) * (1.0 - sigmoid(y)).
```

```
 The way we implemented this requires that the input y is already
sigmoided
 """
 return y * (1-y)
x = np.linspace(-5., 5., num=100)
sig = sigmoid(x)
sig_prime = sigmoid_(sig)
print(np.max(sig_prime))
plt.plot(x, sig, label="sigmoid")
plt.plot(x, sig_prime, label="sigmoid prime")
plt.xlabel("x")
plt.ylabel("y")
plt.legend(prop={'size' : 16})
plt.show()
[ ]
def dense(inputs, weights):
 """A simple dense layer."""
 return np.matmul(inputs, weights)
input_size = 2
hidden_size = 3
output_size = 1
def initialize_weights():
 # weights for hidden layer, shape: 2x3
 w1 = np.random.uniform(size=(input_size, hidden_size))
 # weights for output layer, shape: 3x1
 w2 = np.random.uniform(size=(hidden_size, output_size))
 return w1, w2

w1, w2 = initialize_weights()

def forward_pass(X):
 # Step 1: Calculate weighted average of inputs (output shape: 4x3)
 net_hidden = dense(X, w1)

 # Step 2: Calculate the result of the sigmoid activation function (shape:
4x3)
 act_hidden = sigmoid(net_hidden)
# Step 3: Calculate output of neural network (output shape: 4x1)
 y_hat = dense(act_hidden, w2)

[ ]
def mse(y_hat, y):
 residual = y_hat - y
 error = np.mean(0.5 * (residual ** 2))
 return
```

OUTPUT:


1.0
0.8
0.6
0.4
0.2
0.0
x2
0.0
0.2
0.4
0.6
0.8
1.0
x1
0.2498406452423726


sigmoid
sigmoid prime
1.0
0.8
0.6
0.4
0.2
0.0
y
−4
−2
0
2
4
x