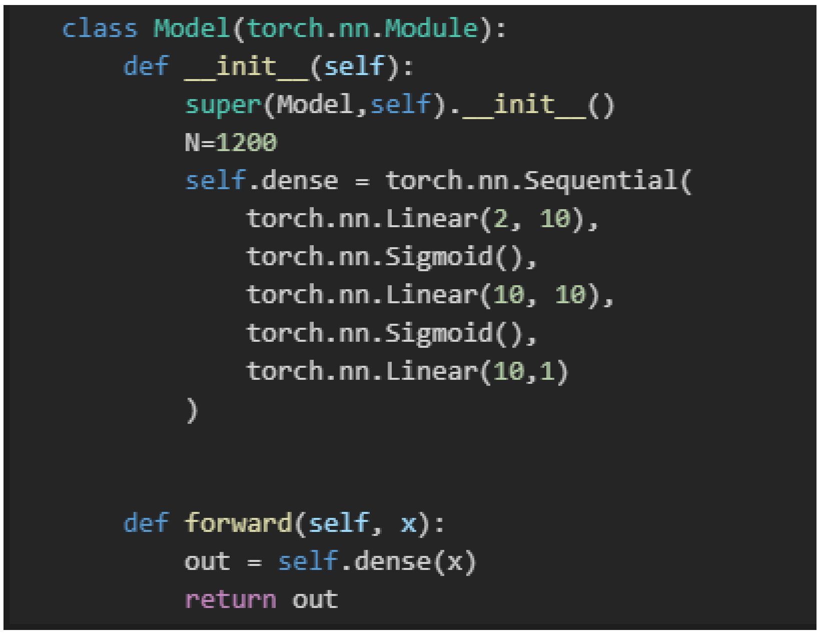
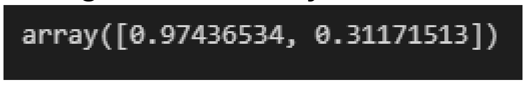
1. The model is built as below:



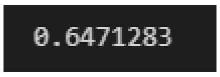
```
class Model(torch.nn.Module):
    def __init__(self):
        super(Model,self).__init__()
        N=1200
        self.dense = torch.nn.Sequential(
            torch.nn.Linear(2, 10),
            torch.nn.Sigmoid(),
            torch.nn.Linear(10, 10),
            torch.nn.Sigmoid(),
            torch.nn.Linear(10,1)
        )


    def forward(self, x):
        out = self.dense(x)
        return out
```

2. Using the randomly drawn X1 & X2 with value as blew:



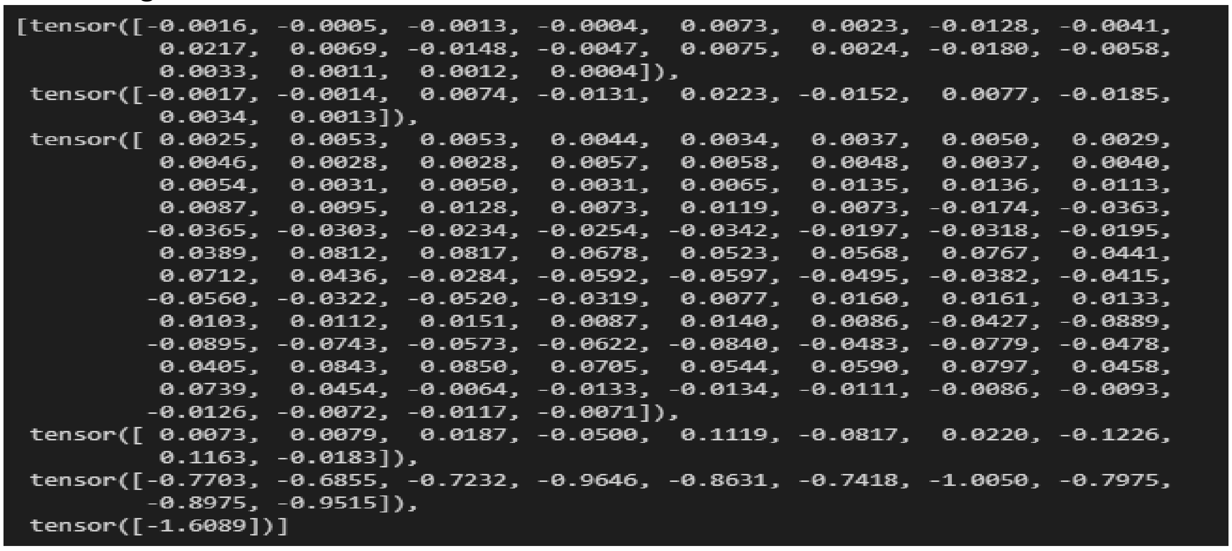
```
array([0.97436534, 0.31171513])
```

The loss value is:



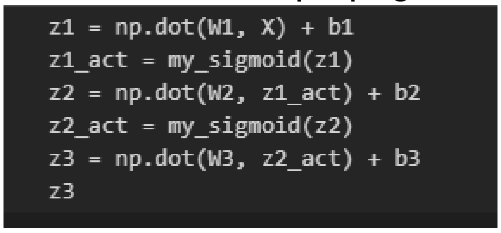
```
0.6471283
```

3. Get the grads of the model:



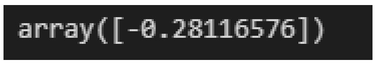
```
[tensor([-0.0016, -0.0005, -0.0013, -0.0004,  0.0073,  0.0023, -0.0128, -0.0041,
          0.0217,  0.0069, -0.0148, -0.0047,  0.0075,  0.0024, -0.0180, -0.0058,
          0.0033,  0.0011,  0.0012,  0.0004]),
 tensor([-0.0017, -0.0014,  0.0074, -0.0131,  0.0223, -0.0152,  0.0077, -0.0185,
          0.0034,  0.0013]),
 tensor([ 0.0025,  0.0053,  0.0053,  0.0044,  0.0034,  0.0037,  0.0050,  0.0029,
          0.0046,  0.0028,  0.0028,  0.0057,  0.0058,  0.0048,  0.0037,  0.0040,
          0.0054,  0.0031,  0.0050,  0.0031,  0.0065,  0.0135,  0.0136,  0.0113,
          0.0087,  0.0095,  0.0128,  0.0073,  0.0119,  0.0073, -0.0174, -0.0363,
         -0.0365, -0.0303, -0.0234, -0.0254, -0.0342, -0.0197, -0.0318, -0.0195,
          0.0389,  0.0812,  0.0817,  0.0678,  0.0523,  0.0568,  0.0767,  0.0441,
          0.0712,  0.0436, -0.0284, -0.0592, -0.0597, -0.0495, -0.0382, -0.0415,
         -0.0560, -0.0322, -0.0520, -0.0319,  0.0077,  0.0160,  0.0161,  0.0133,
          0.0103,  0.0112,  0.0151,  0.0087,  0.0140,  0.0086, -0.0427, -0.0889,
         -0.0895, -0.0743, -0.0573, -0.0622, -0.0840, -0.0483, -0.0779, -0.0478,
          0.0405,  0.0843,  0.0850,  0.0705,  0.0544,  0.0590,  0.0797,  0.0458,
          0.0739,  0.0454, -0.0064, -0.0133, -0.0134, -0.0111, -0.0086, -0.0093,
         -0.0126, -0.0072, -0.0117, -0.0071]),
 tensor([ 0.0073,  0.0079,  0.0187, -0.0500,  0.1119, -0.0817,  0.0220, -0.1226,
          0.1163, -0.0183]),
 tensor([-0.7703, -0.6855, -0.7232, -0.9646, -0.8631, -0.7418, -1.0050, -0.7975,
         -0.8975, -0.9515]),
 tensor([-1.6089])]
```

4. Build the forward propagation in scratch:



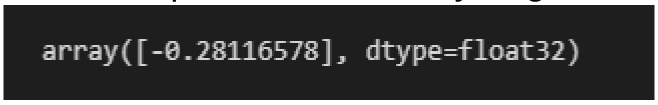
```
z1 = np.dot(W1, X) + b1
z1_act = my_sigmoid(z1)
z2 = np.dot(W2, z1_act) + b2
z2_act = my_sigmoid(z2)
z3 = np.dot(W3, z2_act) + b3
z3
```

The predicted value here is:



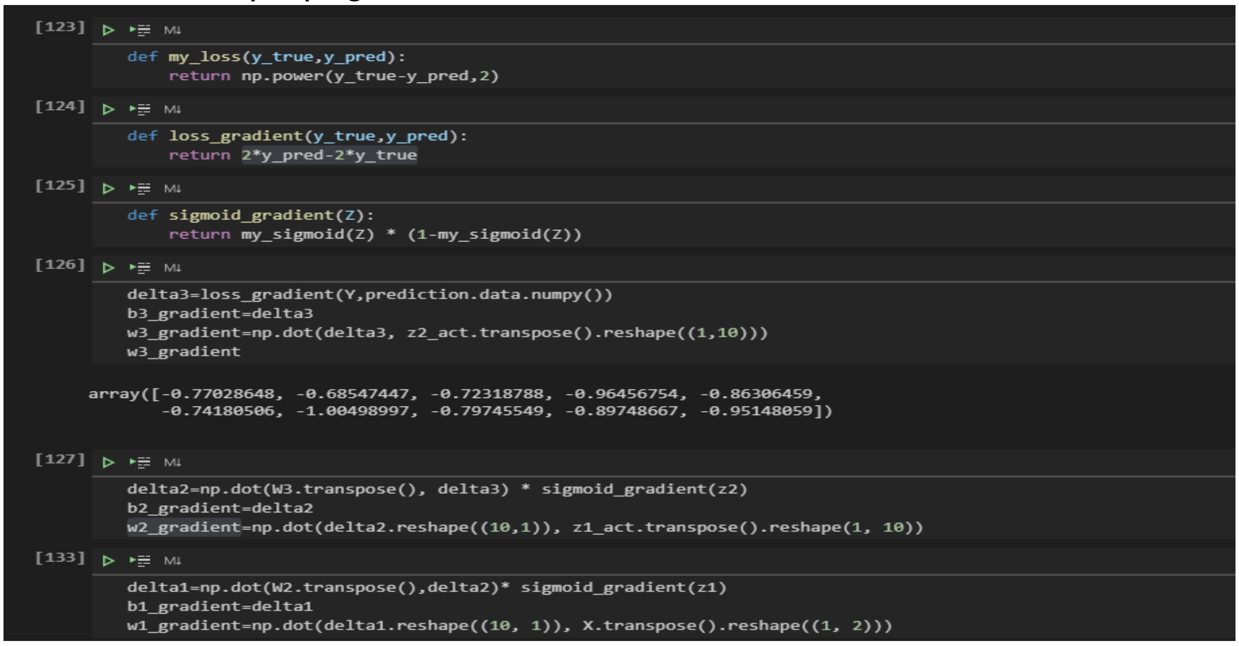
```
array([-0.28116576])
```

While the predicted value by original model is:



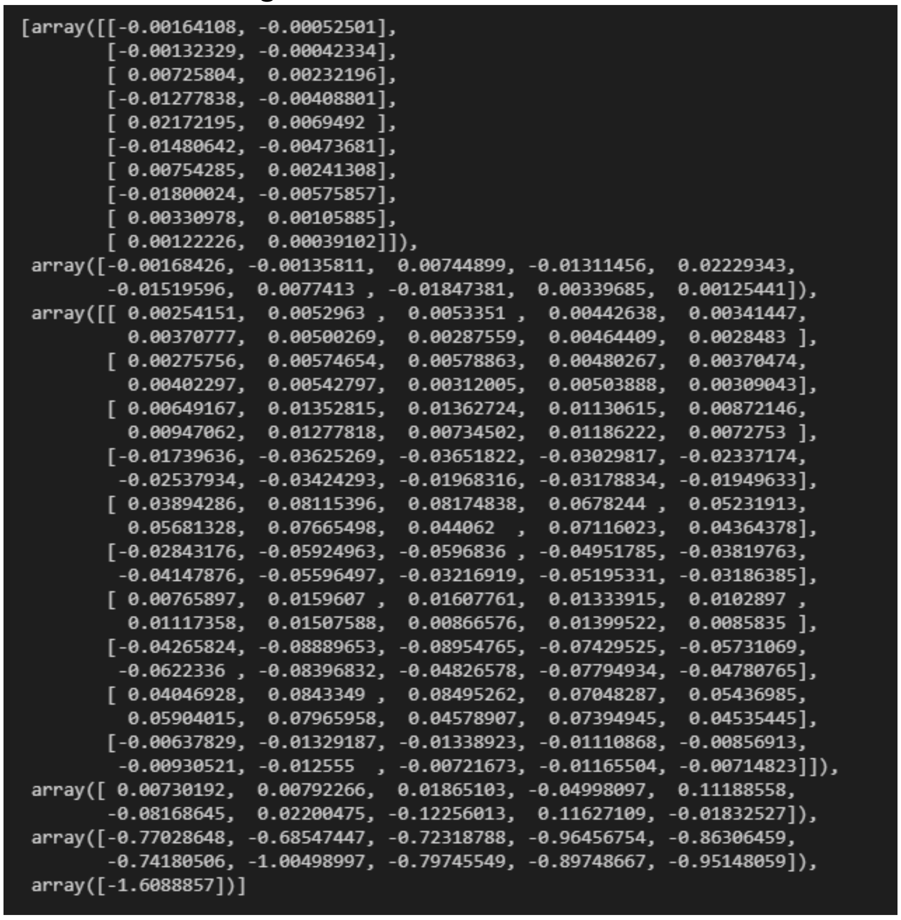
```
array([-0.28116578], dtype=float32)
```

5. Build the back propagation in scratch:



```
[123]
def my_loss(y_true,y_pred):
    return np.power(y_true-y_pred,2)

[124]
def loss_gradient(y_true,y_pred):
    return 2*y_pred-2*y_true

[125]
def sigmoid_gradient(Z):
    return my_sigmoid(Z) * (1-my_sigmoid(Z))

[126]
delta3=loss_gradient(Y,prediction.data.numpy())
b3_gradient=delta3
w3_gradient=np.dot(delta3, z2_act.transpose().reshape((1,10)))
w3_gradient

array([-0.77028648, -0.68547447, -0.72318788, -0.96456754, -0.86306459,
       -0.74180506, -1.00498997, -0.79745549, -0.89748667, -0.95148059])

[127]
delta2=np.dot(W3.transpose(), delta3) * sigmoid_gradient(z2)
b2_gradient=delta2
w2_gradient=np.dot(delta2.reshape((10,1)), z1_act.transpose().reshape(1, 10))

[133]
delta1=np.dot(W2.transpose(),delta2)* sigmoid_gradient(z1)
b1_gradient=delta1
w1_gradient=np.dot(delta1.reshape((10, 1)), X.transpose().reshape((1, 2)))
```

The calculated gradients:



```
[array([[-0.00164108, -0.00052501],
        [-0.00132329, -0.00042334],
        [ 0.00725804,  0.00232196],
        [-0.01277838, -0.00408801],
        [ 0.02172195,  0.0069492 ],
        [-0.01480642, -0.00473681],
        [ 0.00754285,  0.00241308],
        [-0.01800024, -0.00575857],
        [ 0.00330978,  0.00105885],
        [ 0.00122226,  0.00039102]]),
 array([-0.00168426, -0.00135811,  0.00744899, -0.01311456,  0.02229343,
        -0.01519596,  0.0077413 , -0.01847381,  0.00339685,  0.00125441]),
 array([[ 0.00254151,  0.0052963 ,  0.0053351 ,  0.00442638,  0.00341447,
          0.00370777,  0.00500269,  0.00287559,  0.00464409,  0.0028483 ],
        [ 0.00275756,  0.00574654,  0.00578863,  0.00480267,  0.00370474,
          0.00402297,  0.00542797,  0.00312005,  0.00503888,  0.00309043],
        [ 0.00649167,  0.01352815,  0.01362724,  0.01130615,  0.00872146,
          0.00947062,  0.01277818,  0.00734502,  0.01186222,  0.0072753 ],
        [-0.01739636, -0.03625269, -0.03651822, -0.03029817, -0.02337174,
         -0.02537934, -0.03424293, -0.01968316, -0.03178834, -0.01949633],
        [ 0.03894286,  0.08115396,  0.08174838,  0.0678244 ,  0.05231913,
          0.05681328,  0.07665498,  0.044062  ,  0.07116023,  0.04364378],
        [-0.02843176, -0.05924963, -0.0596836 , -0.04951785, -0.03819763,
         -0.04147876, -0.05596497, -0.03216919, -0.05195331, -0.03186385],
        [ 0.00765897,  0.0159607 ,  0.01607761,  0.01333915,  0.0102897 ,
          0.01117358,  0.01507588,  0.00866576,  0.01399522,  0.0085835 ],
        [-0.04265824, -0.08889653, -0.08954765, -0.07429525, -0.05731069,
         -0.0622336 , -0.08396832, -0.04826578, -0.07794934, -0.04780765],
        [ 0.04046928,  0.0843349 ,  0.08495262,  0.07048287,  0.05436985,
          0.05904015,  0.07965958,  0.04578907,  0.07394945,  0.04535445],
        [-0.00637829, -0.01329187, -0.01338923, -0.01110868, -0.00856913,
         -0.00930521, -0.012555  , -0.00721673, -0.01165504, -0.00714823]]),
 array([ 0.00730192,  0.00792266,  0.01865103, -0.04998097,  0.11188558,
        -0.08168645,  0.02200475, -0.12256013,  0.11627109, -0.01832527]),
 array([-0.77028648, -0.68547447, -0.72318788, -0.96456754, -0.86306459,
        -0.74180506, -1.00498997, -0.79745549, -0.89748667, -0.95148059]),
 array([-1.6088857])]
```