Assignment 4 (Li Yonghao)

1. I build the neural network in pytorch by Sequential( ) function, we can add any layers and active function in Sequential( ) to enhance our model.

```
net1 = torch.nn.Sequential(
    torch.nn.Linear(2,10),
    torch.nn.Sigmoid(),
    torch.nn.Linear(10,10),
    torch.nn.Sigmoid(),
    torch.nn.Linear(10,1)
)
```

```
print(net1)
```

```
Sequential(
  (0): Linear(in_features=2, out_features=10, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=10, out_features=10, bias=True)
  (3): Sigmoid()
  (4): Linear(in_features=10, out_features=1, bias=True)
)
```

2. I generated the input data which has 100 samples and 2 dimensions, with the uniform random distribution.

```
sampler = torch.distributions.Uniform(low=0, high=1)
x = sampler.sample((100,2))
x
```

```
tensor([[0.9861, 0.6680],
        [0.4782, 0.7114],
        [0.2804, 0.0288],
        [0.4026, 0.5945],
        [0.7839, 0.0216],
        [0.0385, 0.6850],
        [0.4125, 0.2726],
        [0.1644, 0.9238],
        [0.1199, 0.6362],
        [0.0870, 0.0672],
```

3. I generate the labels y = (x1*x1+x2*x2)/2.

```
y = (x[:,0]*x[:,0] + x[:,1]*x[:,1])/2
y
```

```
tensor([0.7093, 0.3673, 0.0397, 0.2578, 0.3075, 0.2353, 0.1222, 0.4403, 0.2096,
        0.0060, 0.9020, 0.6275, 0.4476, 0.1032, 0.2235, 0.4922, 0.5116, 0.4026,
        0.7083, 0.4988, 0.1050, 0.3979, 0.4567, 0.1443, 0.4335, 0.1293, 0.5482,
        0.3703, 0.4209, 0.2701, 0.3060, 0.6231, 0.5718, 0.0031, 0.4985, 0.3455,
        0.0698, 0.4905, 0.0999, 0.1578, 0.4592, 0.1555, 0.7750, 0.2690, 0.4355,
        0.5053, 0.2573, 0.4590, 0.2837, 0.4448, 0.4617, 0.2442, 0.0833, 0.6665,
        0.1941, 0.2009, 0.4391, 0.6247, 0.3877, 0.1557, 0.4815, 0.1875, 0.7758,
        0.0506, 0.4030, 0.4244, 0.3745, 0.0294, 0.4488, 0.2277, 0.2076, 0.4592,
        0.5623, 0.1183, 0.5104, 0.2053, 0.6701, 0.4316, 0.4149, 0.2391, 0.2069,
        0.2090, 0.3632, 0.3943, 0.5326, 0.8041, 0.2750, 0.4379, 0.2567, 0.5691,
        0.3395, 0.3953, 0.2916, 0.8822, 0.2686, 0.8692, 0.2299, 0.1073, 0.4933,
        0.1096])
```

4. Implement a loss function L = (predict-y)^2.

```
def L(predict,y):
    return torch.sum((predict-y)**2)
```

5. Use batch size of 1 to do one time forward propagation with one data point. lr is the learning rate, I choose 0.05, then construct an optimizer object, and I call my lost function to check if my model is successful.

```
x[0]
```

```
tensor([0.9861, 0.6680])
```

```
y[0]
```

```
tensor(0.7093)
```

```
optimizer = torch.optim.SGD(net1.parameters(), lr=0.05)
optimizer.zero_grad()
prediction = net1(x[0])
loss = L(prediction, y[0])
loss.backward()
print("Loss = " + str(loss.data.numpy()))
```

```
Loss = 0.025985185
```

6. Compute the gradients using pytorch autograd, the result is below:

```
Loss = 0.68349
y_prediction:[-0.72003]
Input_layer w_gradient: [[-1.00e-05 -8.00e-05]
 [ 2.10e-04  1.13e-03]
 [ 9.00e-05  5.00e-04]
 [-1.30e-04 -6.80e-04]
 [-2.80e-04 -1.52e-03]
 [ 6.80e-04  3.67e-03]
 [-1.13e-03 -6.05e-03]
 [ 2.10e-04  1.12e-03]
 [ 1.10e-03  5.93e-03]
 [ 9.00e-04  4.85e-03]]
Input_layer b_gradient: [-0.00017  0.00248  0.0011  -0.0015  -0.00336  0.00809 -0.01333  0.00246
  0.01305  0.01068]
Hidden_layer_1 weight_gradient: [[ 0.02321  0.02342  0.0252   0.03736  0.03658  0.03411  0.0398   0.02897
   0.02532  0.01899]
 [ 0.02352  0.02373  0.02553  0.03786  0.03706  0.03456  0.04032  0.02935
   0.02565  0.01924]
 [ 0.03394  0.03424  0.03684  0.05463  0.05348  0.04988  0.05819  0.04236
   0.03702  0.02776]
 [ 0.0408   0.04117  0.0443   0.06568  0.0643   0.05997  0.06997  0.05093
   0.04451  0.03338]
 [ 0.0046   0.00464  0.00499  0.0074   0.00724  0.00675  0.00788  0.00574
   0.00501  0.00376]
 [-0.01027 -0.01036 -0.01115 -0.01653 -0.01618 -0.01509 -0.01761 -0.01282
  -0.0112  -0.0084 ]
 [ 0.03103  0.03131  0.03368  0.04995  0.0489   0.0456   0.0532   0.03873
   0.03385  0.02538]
 [-0.01203 -0.01214 -0.01307 -0.01937 -0.01897 -0.01769 -0.02064 -0.01502
  -0.01313 -0.00985]
 [-0.0257  -0.02594 -0.02791 -0.04138 -0.04051 -0.03778 -0.04408 -0.03208
  -0.02804 -0.02103]
 [ 0.02161  0.0218   0.02346  0.03478  0.03405  0.03176  0.03705  0.02697
   0.02357  0.01768]]
Hidden_layer_1 bias_gradient: [ 0.05976  0.06056  0.08739  0.10507  0.01183 -0.02644  0.07989 -0.03099
 -0.06619  0.05564]
Hidden_layer_2 weight_gradient: [-0.73496 -0.54626 -0.87168 -0.9232  -0.63875 -0.87525 -0.79051 -0.6599
 -0.84924 -1.06119]
Hidden_layer_2 bias_gradient: [-1.65347]
```

7. Implement the forward propagation and backpropagation algorithm from scratch:

Forward

```python
hidden1 = x0.dot(w1)+b1
hidden1_sigmoid = 1.0 / (1.0 + np.exp(-hidden1))
hidden2 = hidden1_sigmoid.dot(w2)+b2
hidden2_sigmoid = 1.0 / (1.0 + np.exp(-hidden2))
prediction_ = hidden2_sigmoid.dot(w3)+b3
loss_forward = np.square(prediction_ - y0).sum()
```

Backward

```python
dy_prediction_ = 2.0 * (prediction_ - y0)

dt = float(dy_prediction_)
dw3 = np.dot(hidden2_sigmoid.T, dt)
db3 = np.ones(1).dot(dt)

dt = np.dot(dt, w3.T)*hidden2_sigmoid*(1-hidden2_sigmoid)
dw2 = np.dot(hidden1_sigmoid.reshape(len(hidden1_sigmoid),1), dt)
db2 = np.ones(1).dot(dt)

dt = np.dot(dt, w2.T)*hidden1_sigmoid*(1-hidden1_sigmoid)
dw1 = np.dot(x0.reshape(len(x0),1), dt)
db1 = np.ones(1).dot(dt)

w1 -= 0.05 * dw1
w2 -= 0.05 * dw2
w3 -= 0.05 * dw3.reshape(len(dw3),1)

b1 -= 0.05 * db1
b2 -= 0.05 * db2
b3 -= 0.05 * db3
```

Forward:

$$\vec{h}_1 = \vec{x} \cdot \vec{w}_1 + \vec{b}_1$$

$$\overrightarrow{h_1\text{-sigmoid}} = \frac{1}{1+e^{-\vec{h}_1}}$$

$$\vec{h}_2 = \overrightarrow{h_1\text{-sigmoid}} \cdot \vec{W}_2 + \vec{b}_2$$

$$\overrightarrow{h_2\text{-sigmoid}} = \frac{1}{1+e^{-\vec{h}_2}}$$

$$\overrightarrow{\text{prediction}} = \overrightarrow{h_2\text{-sigmoid}} \cdot \vec{W}_3 + \vec{b}_3$$

```
Loss = 0.68349
y prediction: [-0.72003]
Input_layer w_gradient: [[-1.00e-05 -8.00e-05]
 [ 2.10e-04  1.13e-03]
 [ 9.00e-05  5.00e-04]
 [-1.30e-04 -6.80e-04]
 [-2.80e-04 -1.52e-03]
 [ 6.80e-04  3.67e-03]
 [-1.13e-03 -6.05e-03]
 [ 2.10e-04  1.12e-03]
 [ 1.10e-03  5.93e-03]
 [ 9.00e-04  4.85e-03]]
Input_layer b_gradient: [-0.00017  0.00248  0.0011  -0.0015  -0.00336  0.00809 -0.01333  0.00246
  0.01305  0.01068]
Hidden_layer_1 weight_gradient: [[ 0.02321  0.02352  0.03394  0.0408   0.0046  -0.01027  0.03103 -0.01203
  -0.0257   0.02161]
 [ 0.02342  0.02373  0.03424  0.04117  0.00464 -0.01036  0.03131 -0.01214
  -0.02594  0.0218 ]
 [ 0.0252   0.02553  0.03684  0.0443   0.00499 -0.01115  0.03368 -0.01307
  -0.02791  0.02346]
 [ 0.03736  0.03786  0.05463  0.06568  0.0074  -0.01653  0.04995 -0.01937
  -0.04138  0.03478]
 [ 0.03658  0.03706  0.05348  0.0643   0.00724 -0.01618  0.0489  -0.01897
  -0.04051  0.03405]
 [ 0.03411  0.03456  0.04988  0.05997  0.00675 -0.01509  0.0456  -0.01769
  -0.03778  0.03176]
 [ 0.0398   0.04032  0.05819  0.06997  0.00788 -0.01761  0.0532  -0.02064
  -0.04408  0.03705]
 [ 0.02897  0.02935  0.04236  0.05093  0.00574 -0.01282  0.03873 -0.01502
  -0.03208  0.02697]
 [ 0.02532  0.02565  0.03702  0.04451  0.00501 -0.0112   0.03385 -0.01313
  -0.02804  0.02357]
 [ 0.01899  0.01924  0.02776  0.03338  0.00376 -0.0084   0.02538 -0.00985
  -0.02103  0.01768]]
Hidden_layer_1 bias_gradient: [ 0.05976  0.06056  0.08739  0.10507  0.01183 -0.02644  0.07989 -0.03099
 -0.06619  0.05564]
Hidden_layer_2 weight_gradient: [-0.73496 -0.54626 -0.87168 -0.9232  -0.63875 -0.87525 -0.79052 -0.6599
 -0.84924 -1.0612 ]
Hidden_layer_2 bias_gradient: [-1.65347]
```

$$\frac{d\,loss}{d z_3} = 2 \cdot (Pre - y)$$

$$\frac{d\,loss}{d w_3} = \frac{d\,loss}{d z_3} \frac{d z_3}{d w_3} = 2 \cdot (Pre - y) \cdot \text{Sigmoid}(z_2)$$

$$\frac{d\,loss}{d w_2} = \frac{d\,loss}{d z_3} \cdot \frac{d z_3}{d z_2} \cdot \frac{d z_2}{d w_2} = 2 \cdot (Pre - y)\left(W_3 \cdot (1 - z_2)(z_2)\right) \text{Sigmoid}(z_1)$$

$$\frac{d\,loss}{d w_1} = \frac{d\,loss}{d z_3} \cdot \frac{d z_3}{d z_2} \cdot \frac{d z_2}{d z_1} \cdot \frac{d z_1}{d w_1} = 2(Pre - y) W_3 \cdot (1 - z_2)(z_2) W_2 (1 - z_1) z_1 \cdot x \;\cancel{\text{Sigmoid}(x)}$$

$$\frac{d\,loss}{d b_3} = \cancel{2 \cdot (Pre - y)}\ \frac{d\,loss}{d z_3} \frac{d z_3}{d b_3} = 2 \cdot (Pre - y) \cdot 1$$

$$\frac{d\,loss}{d b_2} = \frac{d\,loss}{d z_3} \cdot \frac{d z_3}{d z_2} \cdot \frac{d z_2}{d b_2} = 2 \cdot (Pre - y) \cdot W_3 (1 - z_2) \cdot z_2$$

$$\frac{d\,loss}{d b_1} = \frac{d\,loss}{d z_3} \cdot \frac{d z_3}{d z_2} \frac{d z_2}{d z_1} \frac{d z_1}{d b_1} = 2(Pre - y) \cdot W_3 (1 - z_2) z_2 \cdot W_2 (1 - z_1) z_1$$

8. Compare the two results: They are the same.

```python
rint("loss_deviation = " + str(np.round(loss.item(),5) - np.round(loss_forward.item(),5)))
rint("prediction_deviation = " + str(np.round(prediction.item(),5) - np.round(prediction_.item(),5)))
rint("input_layer w_deviation = " + str((np.round(input_layer.weight.grad.tolist(),4)-np.round(dw1.T,4)).sum()))
rint("input_layer b_deviation = " + str((np.round(input_layer.bias.grad.tolist(),4)-np.round(db1,4)).sum()))
rint("hidden_layer_1 w_deviation = " + str((np.round(hidden_layer1.weight.grad.tolist(),4)-np.round(dw2,4)).sum()))
rint("hidden_layer_1 b_deviation = " + str((np.round(hidden_layer1.bias.grad.tolist(),4)-np.round(db2,4)).sum()))
rint("hidden_layer_2 w_deviation = " + str((np.round(hidden_layer2.weight.grad.tolist(),4)-np.round(dw3,4)).sum()))
rint("hidden_layer_2 b_deviation = " + str((np.round(hidden_layer2.bias.grad.tolist(),4)-np.round(db3,4)).sum()))
```

```
loss_deviation = 0.0
prediction_deviation = 0.0
input_layer w_deviation = 0.0
input_layer b_deviation = 0.0
hidden_layer_1 w_deviation = 0.0
hidden_layer_1 b_deviation = 0.0
hidden_layer_2 w_deviation = 0.0
hidden_layer_2 b_deviation = 0.0
```

The github link: https://github.com/YonghaoLi6/Li-Yonghao_Assigament_1