BS6207 Assignment 1

Based on my understanding, this assignment is first testing our mastery of this forward pass and backpropagation of deep learning, especially the mathematical part as well as our skills of Pytorch. The result of this assignment is to calculate gradients of each dL/dw and dL/db with two about methods, and to testify whether they match to each other.

For the first aim, I built the deep learning model with two hidden layers by script. All the packages I used here is math (for building sigmoid method) and numpy (for array calculation).

Algorithm description and corresponding code explanation of the script:

Total model structure:

$$x \xrightarrow{h_1} V_1 \xrightarrow{h_2} V_2 \xrightarrow{h_3} V_3 = O$$

Frist forward pass:

$$z_2 = v_1 * w_2 + b_2$$

10 * 10 1 * 10 10 * 10 10 *1

$$v_2 = \sigma(z_2)$$

1 * 10

$$z_1 = x * w_1 + b_1$$

10 * 2 1 * 2 10 * 2 10 * 1

$$v_1 = \sigma(z_1)$$

1 * 10

```
def sigmoid(x):
  # sigmoid function
  return 1 / (1 + math. exp(-x))
```

$$v_3 = z_3 = v_2 * w_3 + b_3$$

1*10 1 * 10 1* 10 1 * 10

```
# second layer pass
if layer == 2:
  z = x * w + b
  v2 = []
  ze = 0
  for i in range(10):
    for j in range(10):
      ze += z[i][j]
    vv = act(ze)
    v2.append(vv)
  if torch_flag:
    return v2,z
  return np.array(v2),z
```

```
def forward(x,w,b,act,layer,torch_flag = False):
  if layer == 1:
    v1 = []
    z = x * w + b
    ze = 0
    for i in range(10):
      for j in range(2):
        ze += z[i][j]
        vv = act(ze)
      v1.append(vv)
    if torch_flag:
      return v1,z
    # the format of data type should be change w
    return np.array(v1),z
```

```
# third layer pass
if layer == 3:
  if torch_flag:
      zz = 0
      for i in range(10):
        zz += x[i] * w[i] + b[i]
      return zz
  z = x * w + b
  return np.sum(z)
```

Loss calculation:

$$L = (pred - y)^2$$

```
def loss(pred,y):
  # loss function
  L = (pred- y)**2
  return L
```

Gradients calculation:

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial w_3} = \frac{\partial L}{\partial z_3} v_2$$

```
# dL/dw3 = 2(yred-y) * v2
def gradient_w3(v,y,pred):
  dw3 = []
  for i in range(10):
    dw = 2 *v[i]* (pred - y)
    dw3.append(dw)
  return np.array(dw3)
```

$$\frac{\partial L}{\partial b_3} = \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial b_3} = \frac{\partial L}{\partial z_3} * 1$$

```
# dL/db3 = 2(yred-y)
def gradient_b3(y,pred):
  db3 = []
  for i in range(10):
    db = 2 * (pred - y)
    db3.append(db)
  return np.array(db3)
```

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial v_2}\frac{\partial v_2}{\partial z_2}\frac{\partial z_2}{\partial w_2} = \frac{\partial L}{\partial z_3} w_3\, \sigma'(z_2) v_1$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial v_2}\frac{\partial v_2}{\partial z_2}\frac{\partial z_2}{\partial b_2} = \frac{\partial L}{\partial z_3} w_3\, \sigma'(z_2) * 1$$

```
#dL/dw2 = 2(yred-y) * w3*sig(z2) * (1-sig(z2)) * v1
def gradient_w2(v,y,pred,z2,w3):
  dw2 = []
  ze = 0
  sigz2 = []
  for i in range(10):
    for j in range(10):
        ze += z2[i][j]
    sigz = sigmoid(ze) * (1 - sigmoid(ze))
    # save the sig(z2), no need to calculate next time
    sigz2.append(sigz)
    dw = 2 *v[i]* (pred - y) * sigz *w3[i]
    dw2.append(dw)
  return np.array(dw2),np.array(sigz2)
#dL/db2 = 2(yred-y) * w3*sig(z2) * (1-sig(z2))
def gradient_b2(y,pred,sigz2,w3):
  db2 = []
  for i in range(10):
    db = 2 * (pred - y)* sigz2[i] *w3[i]
    db2.append([db])
  return np.array(db2)
```

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial v_2}\frac{\partial v_2}{\partial z_2}\frac{\partial z_2}{\partial v_1}\frac{\partial v_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} = \frac{\partial L}{\partial z_3} w_3\, \sigma'(z_2)\, w_2\, \sigma'(z_1) x$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial v_2}\frac{\partial v_2}{\partial z_2}\frac{\partial z_2}{\partial v_1}\frac{\partial v_1}{\partial z_1}\frac{\partial z_1}{\partial b_1} = \frac{\partial L}{\partial z_3} w_3\, \sigma'(z_2)\, w_2\, \sigma'(z_1) * 1$$

```
# dL/dw1 = 2 * (yred - y)* w3 *sig(z2) * w2 * sig(z1) * x
def gradient_w1(x,y,w2,w3,z1,pred,sigz2):
  dw1 = []
  sigz1 = []
  for i in range(10):
    sigz = sigmoid(z1[i][0] + z1[i][1]) * (1 - sigmoid(z1[i][0] + z1[i][1]))
    sigz1.append(sigz)
    w2sum = np.sum(w2[i])
    dwtemp1 = 2 * x[0] * (pred - y) * w3[i] * sigz2[i] * w2sum * sigz
    dwtemp2 = 2 * x[1] * (pred - y) * w3[i] * sigz2[i] * w2sum * sigz
    dw1.append([dwtemp1,dwtemp2])
  return np.array(dw1),np.array(sigz1)

# dL/db1 = 2 * (yred - y)* w3 *sig(z2) * w2 * sig(z1)
def gradient_b1(y,pred,sigz2,sigz1,w2):
  db1 = []
  for i in range(10):
    w2sum = np.sum(w2[i])
    db = 2 * (pred - y)*w3[i]*sigz2[i] * w2sum * sigz1[i]
    db1.append([db])
  return np.array(db1)
```

Code explanation of the gradients calculation by tourch.autograd:

To compare the results of gradient calculation of two methods, I didn’t use linear model which is built in the Pytorch. Because I don’t know how to pass the same parameters as my script into the linear model. Alternatively, I calculated the gradience with following steps:

1. I converted the same input and parameters(w,b) as in script into the tensors, which can be further calculated by Pytorch.
2. I used the same functions in the script to proceed forward pass in which the nn.Sigmoid is used as the active function to the format of tensors.
3. I also used the same loss function to calculated loss. Then I called Loss. backward() function to compile and generate the gradiences.
4. After backward methods, all the gradiences are generated inside each tensors, and then I just printed them by calling autograd.

```
import torch
x_train = torch.tensor(x,requires_grad=True)
y_train = torch.tensor(y,requires_grad=True)

tw1 = torch.tensor(w1, requires_grad=True)
tb1 = torch.tensor(b1, requires_grad=True)
tw2 = torch.tensor(w2, requires_grad=True)
tb2 = torch.tensor(b2, requires_grad=True)
tw3 = torch.tensor(w3, requires_grad=True)
tb3 = torch.tensor(b3, requires_grad=True)

# active function
m = torch.nn.Sigmoid()
# forward
v1,z = forward(x_train,tw1,tb1,m,1,True)
tv1 = torch.stack(v1)
v2,z = forward(tv1,tw2,tb2,m,2,True)
tpred = forward(v2,tw3,tb3,m,3,True)
# loss
L = loss(tpred,y_train)
# backward
L.backward()

print('The predicted y is:')
print(tpred)
print('-------------dw3------------')
print(tw3.grad)
print('-------------db3------------')
print(tb3.grad)
print('-------------dw2------------')
print(tw2.grad)
```

Results and Explanation :

```
The predicted y is: 8.36
```

```
The predicted y is:
tensor(8.3631, dtype=torch.float64, grad_fn=<AddBackward0>)
```

```
------------dw3-----------
[15.58998543 15.59228346 15.59228351 15.59228351 15.59228351 15.59228351
 15.59228351 15.59228351 15.59228351 15.59228351]
------------db3-----------
[15.59228351 15.59228351 15.59228351 15.59228351 15.59228351 15.59228351
 15.59228351 15.59228351 15.59228351 15.59228351]
```

```
------------dw3-----------
tensor([15.5900, 15.5923, 15.5923, 15.5923, 15.5923, 15.5923, 15.5923, 15.5923,
        15.5923, 15.5923], dtype=torch.float64)
------------db3-----------
tensor([15.5923, 15.5923, 15.5923, 15.5923, 15.5923, 15.5923, 15.5923, 15.5923,
        15.5923, 15.5923], dtype=torch.float64)
```

From the comparation of the results, we can see the predicted y, dL/dw3, dL/db3 and dL/db2 are the same in two methods.

However, the dimension of the dL/dw2 is different in two methods. The results of torch is 10 * 10, whereas the gradients in my script is 1*10. I checked the equation.

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial v_2}\frac{\partial v_2}{\partial z_2}\frac{\partial z_2}{\partial w_2}$$

$$= \frac{\partial L}{\partial z_3} w_3\, \sigma'(z_2) v_1$$

```
-------------dw2------------
[8.45728103e-04 4.84804552e-08 5.46632872e-16 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00]
-------------db2------------
[[9.15226727e-04]
 [4.88109015e-08]
 [5.47253208e-16]
 [0.00000000e+00]
 [0.00000000e+00]
 [0.00000000e+00]
 [0.00000000e+00]
 [0.00000000e+00]
 [0.00000000e+00]
 [0.00000000e+00]]
```

```
-------------db2-------
tensor([[9.1528e-03],
        [4.8811e-07],
        [5.4725e-15],
        [0.0000e+00],
        [0.0000e+00],
        [0.0000e+00],
        [0.0000e+00],
        [0.0000e+00],
        [0.0000e+00],
        [0.0000e+00]],
```

It turned out I mistakenly thought the results of $\frac{\partial L}{\partial z_{3i}} w_{3i}$ $\sigma'(z_{2i}) v_{1i}$ are the same for each w_{2i}. Thus, the results were wrong.

Then I found that for the calculation of gradients, we should use $w_3.T$ as the multiplier rather than w_3. After I corrected my error, the most of my results are matched that of torch.auto.

I also built another Linear model with Pytorch to generate gradients, which is in my code. However, the parameter is different from my script, thus it impossible to do comparation.