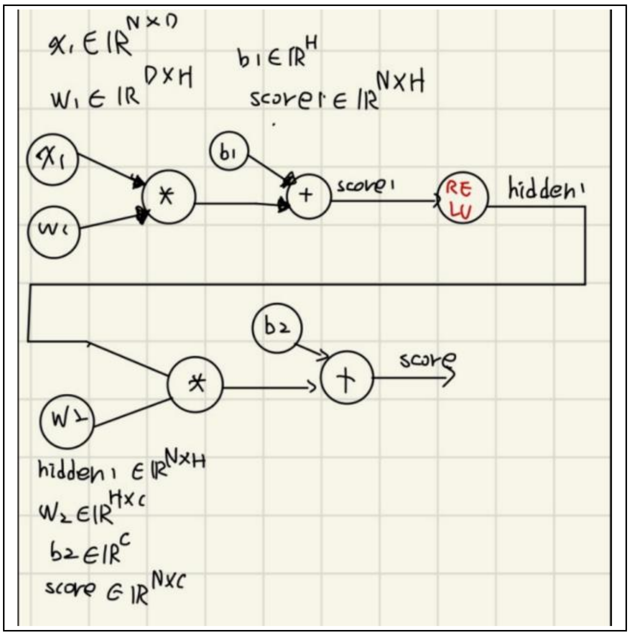
HW5 Two Layer Neural Network(1)

M113040105 劉東霖

壹. nn_forward_pass:

一. 實作過程:

(1). 神經網路的結構圖如下:



(2). 根據下圖的程式碼:

1. 先把 X 和 W1 做矩陣相乘+bias b1 得到 score1

2. 將 score1 進行 relu，把<0 的 score 變成 0，就完成第 1 層的輸出了

3. 將第 1 層的輸出和 W2 做矩陣相乘+bias b2 得到神經網路的輸出

```
score1  =  X  @  W1  +  b1
hidden  =  torch.max(torch.zeros_like(score1),score1)
score2  =  hidden  @  W2  +  b2
scores  =  score2
```

二. 執行結果:

(1). 把 toy 的資料帶進去神經網路計算 scores，發現跟正確結果差異不大。

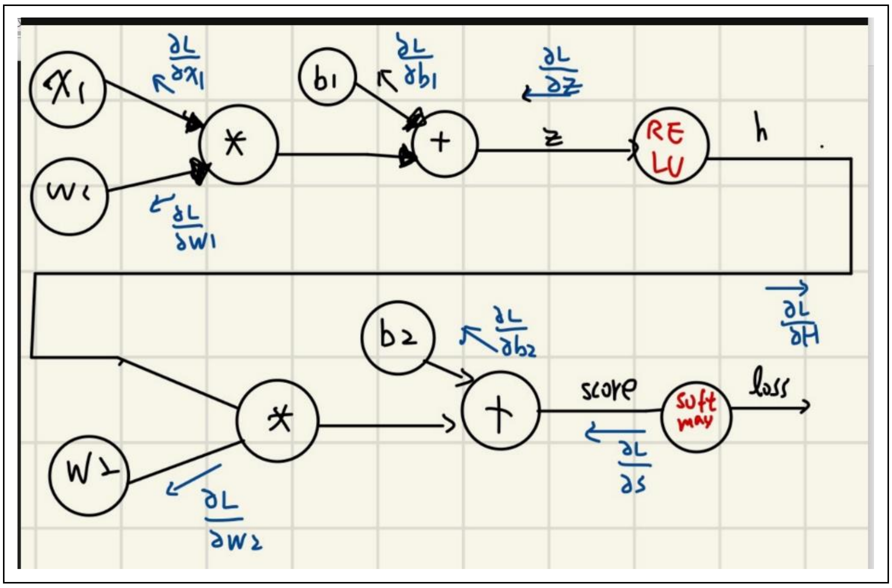
```
Your scores:
tensor([[ 9.7003e-08, -1.1143e-07, -3.9961e-08],
        [-7.4297e-08,  1.1502e-07,  1.5685e-07],
        [-2.5860e-07,  2.2765e-07,  3.2453e-07],
        [-4.7257e-07,  9.0935e-07,  4.0368e-07],
        [-1.8395e-07,  7.9303e-08,  6.0360e-07]], device='cuda:0')
torch.float32

correct scores:
tensor([[ 9.7003e-08, -1.1143e-07, -3.9961e-08],
        [-7.4297e-08,  1.1502e-07,  1.5685e-07],
        [-2.5860e-07,  2.2765e-07,  3.2453e-07],
        [-4.7257e-07,  9.0935e-07,  4.0368e-07],
        [-1.8395e-07,  7.9303e-08,  6.0360e-07]], device='cuda:0')

Difference between your scores and correct scores: 2.24e-11
```

貳. nn_forward_backward:

一. 目的:把神經網路輸出的結果計算出 softmax loss，並用 backpropagation 計算出每個神經元的梯度並更新。

二. 實作過程:

(1). Backpropagation 的示意圖如下:



(2). 計算 loss:

1. 減掉 score 裡每一個 row 的最大值，為了避免數值爆炸，程式如下:

```
scores-=torch.max(scores,dim=1,keepdim=True).values
```

2. 利用下圖的方式把每一筆 data 的正確類別找出來，並把 score 指數化。

```
sy=scores[range(N),y]
s_exp=torch.exp(scores)
```

3. 利用下圖公式計算出 loss，並把 loss 除以資料數目再加上 regularization term。跟上一個作業不一樣的地方是，因為此神經網路有兩層，所以有兩個 w 要加。

$$L = \frac{1}{N}\boxed{\sum_i^N L_i} + \lambda \sum w^2$$

$$\sum_i^N -\log\left(\frac{e^{s_i}}{\sum_j e^{s_j}}\right) = \sum_i^N \left(-s_i + \log\left(\sum_j e^{s_j}\right)\right) = -\sum_i^N s_i + \sum_i^N \log\left(\sum_j e^{s_j}\right)$$

程式碼如下:

```
loss=-sum(sy)+torch.sum(torch.log(torch.sum(s_exp,dim=1)))
```

```
loss/=N
loss+=reg*(torch.sum(W1 * W1)+torch.sum(W2 * W2))
```

(3). 計算$\frac{\partial L}{\partial s}$

1. 首先要先計算$\frac{\partial L}{\partial s}$，其中 p 為經過指數化再歸一化過後的機率，再把 p 裡面正確類別的地方等於-1，最後再除以資料數目，計算過程如下:

scalar : $L_i = -\log\left(\frac{e^{s_i}}{\sum_j e^{s_j}}\right) = -s_i + \log\left(\sum_j e^{s_j}\right)$

$$\Rightarrow \frac{\partial L_i}{\partial s_j} = \begin{cases} (P_j - 1) & \text{if } j = y \\ P_j & \text{if } j \neq y \end{cases}$$

Matrix : $S \in \mathbb{R}^{N\times C} \Rightarrow \frac{\partial L}{\partial S} = \begin{cases} P_{i,j} - 1 & \text{if } j = y_i \\ P_{i,j} & \text{if } j \neq y_i \end{cases}$

2. 程式碼如下:

```
p=s_exp/torch.sum(s_exp,dim=1).reshape(-1,1)
p[range(N),y]-=1
```

```
dscore  = p
dscore  /= N
```

(4). 計算$\frac{\partial L}{\partial w_2}$

1. 計算過程如下，其中 H 為第一層神經網路輸出後經過 relu 的結果，$\frac{\partial L}{\partial s}$為 loss 對 score 的微分:

scalar : $\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial s}\frac{\partial s}{\partial W_2} = H\frac{\partial L}{\partial s}$, $S = HW_2 + 1_N b_2 \in \mathbb{R}^{N\times C}$

H×C　　H×N　N×C

Matrix : $W_2 \in \mathbb{R}^{H\times C} \Rightarrow \frac{\partial L}{\partial W_2} = \frac{\partial s}{\partial W_2}\frac{\partial L}{\partial s}$

因 $H \in \mathbb{R}^{N\times H} \Rightarrow \frac{\partial L}{\partial W_2} = H^T \frac{\partial L}{\partial S}$

2. 程式碼如下:

```
dw2  =  h1.t()  @  dscore
```

(5). 計算$\frac{\partial L}{\partial b_2}$

1. 計算過程如下，其中$\frac{\partial L}{\partial s}$為 loss 對 score 的微分，1n 的轉置@$\frac{\partial L}{\partial s}$可以寫成[1, 1, 1, ... , 1] $\frac{\partial L}{\partial s}$，所以$\frac{\partial L}{\partial b_2}$可以寫成$\frac{\partial L}{\partial s}$column的總和:

$$\text{scalar: } \frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial s}\frac{\partial s}{\partial b_2} = \frac{\partial L}{\partial s}, \quad s = HW_2 + 1_N b_2 \in \mathbb{R}^{N\times C}$$

$$\text{Matrix: } b_2 \in \mathbb{R}^{1\times C} \Rightarrow \underset{1\times C}{\frac{\partial L}{\partial b_2}} = \underset{1\times N}{\frac{\partial s}{\partial b_2}} \; \underset{N\times C}{\frac{\partial L}{\partial s}}$$

$$\text{因 } 1_N \in \mathbb{R}^{N\times 1} \Rightarrow \frac{\partial L}{\partial b_2} = 1_N^T \frac{\partial L}{\partial s}$$

$$= [1,1,1,1,\ldots\ldots 1]\frac{\partial L}{\partial s} = \underline{\mathrm{sum}\left(\frac{\partial L}{\partial s}\right)}$$

2. 程式碼如下:

```
db2  =  torch.sum(dscore,dim=0)
```

(6). 計算$\frac{\partial L}{\partial h}$和$\frac{\partial L}{\partial z}$

1. 首先計算$\frac{\partial L}{\partial h}$，計算過程如下，其中$\frac{\partial L}{\partial s}$為 loss 對 score 的微分，w2 為神經網路第二層的權重，此 loss 為經過 relu 後的 loss:

$$\text{Scalar: } \frac{\partial L}{\partial H} = \frac{\partial L}{\partial s}\frac{\partial s}{\partial H} = W_2\frac{\partial L}{\partial s}, \quad s = HW_2 + 1_N b_2 \in \mathbb{R}^{N\times C}$$

$$\text{Matrix: } H \in \mathbb{R}^{N\times H} \Rightarrow \underset{N\times H}{\frac{\partial L}{\partial H}} = \underset{N\times C}{\frac{\partial L}{\partial s}} \; \underset{C\times H}{\frac{\partial s}{\partial H}}$$

$$\text{因 } W_2 \in \mathbb{R}^{H\times C} \Rightarrow \underline{\frac{\partial L}{\partial H} = \frac{\partial L}{\partial s} W_2^T}$$

2. 再來計算$\frac{\partial L}{\partial z}$，計算過程如下，H=relu(z)，此 loss 為經過 relu 前的 loss:

$$\text{scalar: } H: \begin{cases} z, & z>0 \\ 0, & \text{other} \end{cases} \Rightarrow \frac{\partial H}{\partial z} = \begin{cases} 1, & z>0 \\ 0, & \text{otherwise} \end{cases}$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial H}\frac{\partial H}{\partial z} = \begin{cases} \frac{\partial L}{\partial H}, & z>0 \\ 0, & \text{otherwise} \end{cases}$$

$$\text{matrix: } z \in \mathbb{R}^{M\times N} \text{ and } H_{ij} = \begin{cases} z_{ij}, & z_{ij}>0 \\ 0, & \text{otherwise} \end{cases}, \quad \frac{\partial H_{ij}}{\partial z_{ij}} = \begin{cases} 1, & z_{ij}>0 \\ 0, & \text{otherwise} \end{cases}$$

$$\Rightarrow \frac{\partial L}{\partial z} = \frac{\partial L}{\partial H_{ij}}\frac{\partial H_{ij}}{\partial z} = \begin{cases} \frac{\partial L}{\partial H_{ij}}, & z_{ij}>0 \\ 0, & \text{otherwise} \end{cases}$$

3. 程式碼如下:

```
dh1  =  dscore  @  W2.t()
dh1[h1<=0]=0
```

(7). 計算$\frac{\partial L}{\partial w_1}$

1. 計算過程如下，其中 X 為第一層神經網路輸入，$\frac{\partial L}{\partial z}$為 loss 對第一層神經網路輸出的 score 的微分:

$$\text{scalar}: \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial w_1} = \frac{\partial L}{\partial z}x, \quad z = xw_1 + 1^N b_1 \in \mathbb{R}^{N\times H}$$

$$\text{Matrix}: w_1 \in \mathbb{R}^{D\times H} \Rightarrow \overset{D\times H}{\frac{\partial L}{\partial w_1}} = \overset{D\times N}{\frac{\partial z}{\partial w_1}}\ \overset{N\times H}{\frac{\partial L}{\partial z}}$$

$$\text{因 } x \in \mathbb{R}^{N\times D} \Rightarrow \frac{\partial L}{\partial w_1} = \underline{x^T\frac{\partial L}{\partial z}}$$

2. 程式碼如下:

```
dw1  =  X.t()  @  dh1
```

(8). 計算$\frac{\partial L}{\partial b_1}$

1. 計算過程如下，其中$\frac{\partial L}{\partial z}$為 loss 對第一層神經網路的 score 的微分:

$$\text{scalar}: \frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial b_1} = \frac{\partial L}{\partial z}\cdot 1, \quad z = xw_1 + 1^N b_1 \in \mathbb{R}^{N\times H}$$

$$\text{Matrix}: b_1 \in \mathbb{R}^{1\times H} \Rightarrow \underset{1\times H}{\frac{\partial L}{\partial b_1}} = \underset{1\times N}{\frac{\partial z}{\partial b_1}}\ \underset{N\times H}{\frac{\partial L}{\partial z}}$$

$$\text{因 } 1N \in \mathbb{R}^{N\times 1} \Rightarrow \frac{\partial L}{\partial b_1} = 1N^T\frac{\partial L}{\partial z} = [1, 1, \cdots\cdots, 1]\frac{\partial L}{\partial z} = \underline{\mathrm{sum}\left(\frac{\partial L}{\partial z}\right)}$$

2. 程式碼如下:

```
db1  =  torch.sum(dh1, dim=0)
```

(9). 再計算完所有的梯度後，記得在$\frac{\partial L}{\partial w_1}$和$\frac{\partial L}{\partial w_2}$加入 regularization 的梯度，並把算完的梯度存入 grads 的字典裡面，程式如下:

```
dw2  +=  reg  *  W2  *  2
dw1  +=  reg  *  W1  *  2
grads['W1'] = dw1
grads['W2'] = dw2
grads['b1'] = db1
grads['b2'] = db2
```

(10).完整程式如下:

```
scores-=torch.max(scores,dim=1,keepdim=True).values
sy=scores[range(N),y]
s_exp=torch.exp(scores)
loss=-sum(sy)+torch.sum(torch.log(torch.sum(s_exp,dim=1)))
p=s_exp/torch.sum(s_exp,dim=1).reshape(-1,1)
p[range(N),y]-=1
loss/=N
loss+=reg*(torch.sum(W1 * W1)+torch.sum(W2 * W2))
```

```
dscore  =  p
dscore  /=  N
dw2  =  h1.t()  @  dscore
db2  =  torch.sum(dscore,dim=0)
dh1  =  dscore  @  W2.t()
dh1[h1<=0]=0
dw1  =  X.t()  @  dh1
db1  =  torch.sum(dh1,dim=0)
dw2  +=  reg  *  W2  *  2
dw1  +=  reg  *  W1  *  2
grads['W1']  =  dw1
grads['W2']  =  dw2
grads['b1']  =  db1
grads['b2']  =  db2
```

三.執行結果:

1.如下圖所示，我計算出來的 loss 和實際上的 loss 沒有差異:

```
# YOUR_TURN: Implement the loss computation part of nn_forward_backward
loss, _ = nn_forward_backward(params, toy_X, toy_y, reg=0.05)
print('Your loss: ', loss.item())
correct_loss = 1.0986121892929077
print('Correct loss: ', correct_loss)
diff = (correct_loss - loss).item()

# should be very small, we get < 1e-4
print('Difference: %.4e' % diff)
```

```
Your loss:  1.0986121892929077
Correct loss:  1.0986121892929077
Difference: 0.0000e+00
```

2. 如下圖所示，計算出來的$\frac{\partial L}{\partial b_1}$和$\frac{\partial L}{\partial b_2}$和$\frac{\partial L}{\partial w_1}$和$\frac{\partial L}{\partial w_2}$和用 numeric gradient 算出來結果的差異很小:

```
loss, grads = nn_forward_backward(params, toy_X, toy_y, reg=reg)

for param_name, grad in grads.items():
    param = params[param_name]
    f = lambda w: nn_forward_backward(params, toy_X, toy_y, reg=reg)[0]
    grad_numeric = usefuns.grad.compute_numeric_gradient(f, param)
    error = usefuns.grad.rel_error(grad, grad_numeric)
    print('%s max relative error: %e' % (param_name, error))
```

```
W1 max relative error: 1.764326e-06
W2 max relative error: 1.856848e-06
b1 max relative error: 8.548088e-06
b2 max relative error: 1.400721e-09
```

參.nn_train 和 nn_predict:

一.nn_train 實作過程:

(1).題目只要我們更新參數而已，所以我就把各自的參數取出來，減掉(梯度乘學習率)，再儲存回字典。程式碼如下:

```
W1, b1 = params['W1'], params['b1']
W2, b2 = params['W2'], params['b2']
W1 -= learning_rate * grads['W1']
W2 -= learning_rate * grads['W2']
b1 -= learning_rate * grads['b1']
b2 -= learning_rate * grads['b2']
params['W1'], params['b1'] = W1, b1
params['W2'], params['b2'] = W2, b2
```

二. nn_predict 實作過程:

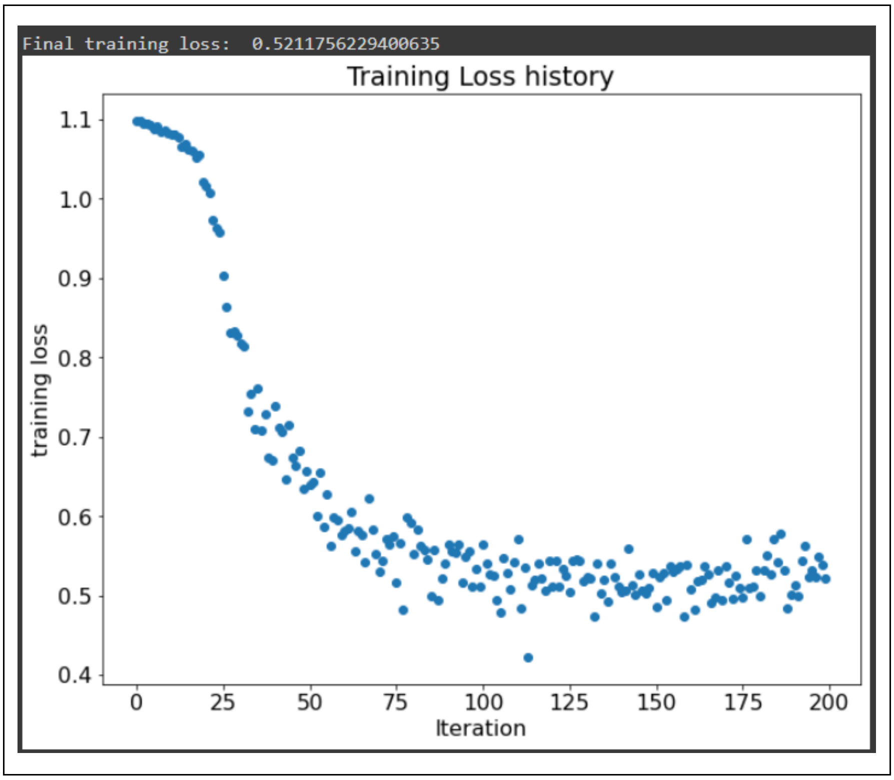
(1). 先把要預測的東西帶入神經網路算出 score，再把每一個 row 裡面分數最高的用 torch.max 取出來，程式碼如下:

```
W1, b1 = params['W1'], params['b1']
W2, b2 = params['W2'], params['b2']
# Compute the forward pass
score1 = X @ W1 + b1
hidden = torch.max(torch.zeros_like(score1),score1)
score2 = hidden @ W2 + b2
scores = score2
_,y_pred=torch.max(scores,dim=1)
```

三. 執行結果:

(1). 把資料帶入 nn_train 裡去訓練，發現隨著 epoch 越多，loss 下降越多，到了某一個 epoch 時開始飽和了:

```
stats = nn_train(params, nn_forward_backward, nn_predict, toy_X, toy_y, toy_X, toy_y,
                 learning_rate=1e-1, reg=1e-6,
                 num_iters=200, verbose=False)
```



(2). 如下圖所示，準確率的部分，也隨著 epoch 越多而增加: