Hanze Yao
3033093286

CS189 HW6

2. Linear Neural Networks

(a) ① If there exists $W_1, W_2, \dots, W_L$ such that $\bar{W} = W_0$ then $\text{rank}(W_0) \le \min_{i=0}^{L} d_i$.

Proof: $\because W_0 = \bar{W} = W_L \times W_{L-1} \times \dots \times W_1$

$\therefore \text{rank}(W_0) = \min(\text{rank}(W_L), \text{rank}(W_{L-1}), \dots, \text{rank}(W_1))$

$\because \text{rank}(W_L) \le \min(d_L, d_{L-1})$

$\text{rank}(W_{L-1}) \le \min(d_{L-1}, d_{L-2})$

$\vdots$

$\text{rank}(W_1) \le \min(d_1, d_0)$

$\therefore \text{rank}(W_0) \le \min_{i=0}^{L} d_i$

② If $\text{rank}(W_0) \le \min_{i=0}^{L} d_i$ then there exists $W_1, W_2, \dots, W_L$ such that $\bar{W} = W_0$

Proof: Assume $\text{rank}(W_0) = d^* \le \min_{i=0}^{L} d_i$

$\text{rank}(W_i) = R_i$

$$W_0 = \underset{p\times p}{U_0} \underset{p \times d}{\begin{pmatrix} s_1 & & & 0 \\ & \ddots & & \\ & & s_{d^*} & \\ 0 & & & 0 \end{pmatrix}} \underset{d\times d}{V_0'}$$

$$\bar{W} = W_L \times W_{L-1} \times \dots \times W_1$$

$$= \underset{p\times p}{U_L} \underset{p\times d_{L-1}}{\begin{pmatrix} s_1 & & & 0 \\ & \ddots & & \\ & & s_{R_L} & \\ 0 & & & 0 \end{pmatrix}} \underbrace{\left(\underset{d_{L-1}\times d_{L-1}}{V_L} \times \underset{d_{L-1}\times d_{L-1}}{U_{L-1}}\right)}_{= I_{d_{L-1}}} \begin{pmatrix} s_1 & & & 0 \\ & \ddots & & \\ & & s_{R_{L-1}} & \\ 0 & & & 0 \end{pmatrix} \underbrace{\left(V_{L-1}' \times U_{L-2}\right)}_{= I_{d_{L-2}}} \cdots$$

$$\cdots \underbrace{\left(V_2' \times U_1\right)}_{= I_{d_1}} \begin{pmatrix} s_1 & & & 0 \\ & \ddots & & \\ & & s_{R_1} & \\ 0 & & & 0 \end{pmatrix} V_1'$$

$$= \underset{p\times p}{U_L} \underbrace{\begin{pmatrix} s_1 & & 0 \\ & \ddots \ s_{R_L} & \\ 0 & & 0 \end{pmatrix} \begin{pmatrix} s_1 & & 0 \\ & \ddots \ s_{R_{L-1}} & \\ 0 & & 0 \end{pmatrix} \cdots \begin{pmatrix} s_1 & & 0 \\ & \ddots \ s_{R_1} & \\ 0 & & 0 \end{pmatrix}}_{p\times d} \underset{d\times d}{V_1'}$$

$$= \underset{p\times p}{U_L} \begin{bmatrix} s_1 & & & 0 \\ & \ddots & & \\ & & s_{d'} & \\ 0 & & & 0 \end{bmatrix} \underset{d\times d}{V_1'}$$

$$d' = \min_i^L (R_i) \le \min_i^L d_i$$

$$\therefore d^* \le \min_i^L d_i$$

$\therefore$ d' can be d^*

$\therefore$ $\bar{W}$ can be equal to W_0

(b)

$$\hat{R}(f_{W_{1:L}}) = \frac{1}{2}\sum_{i=1}^{n} \| f_{W_{1:L}}(x_i) - y_i \|_2^2$$

$$= \frac{1}{2}\sum_{i=1}^{n} \| \bar{W} x_i - y_i \|_2^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p} \| \bar{W}[j] x_i - y_i \|_2^2$$

$$= \frac{1}{2} \| \bar{W} X^T - Y^T \|_F^2 = Q(\bar{W})$$

$$Q(\bar{W}) = \frac{1}{2} \| \bar{W} X^T - Y^T \|_F^2$$

$$= \sum_{j=1}^{p} \frac{1}{2} \| \bar{W}[j] X^T - Y^T[j] \|_2^2$$

$$\nabla_{\bar{W}[j]} \left(\frac{1}{2} \| \bar{W}[j] X^T - Y^T[j] \|_2^2 \right)$$

$$= \frac{1}{2} \nabla_{\bar{W}[j]} \left[(\bar{W}[j] X^T - Y^T[j]) (\bar{W}[j] X^T - Y^T[j])^T \right]$$

$$= \frac{1}{2} \nabla_{\bar{W}[j]} \left(\bar{W}[j] X^T X \bar{W}[j]^T - 2 \bar{W}[j] X^T Y[j] + Y^T[j] Y[j] \right)$$

$$= \frac{1}{2} \left(2 X^T X \bar{W}[j]^T - 2 X^T Y[j] \right)$$

$$= X^T X \bar{W}[j]^T - X^T Y[j]$$

$$\nabla^2_{\bar{W}[j]} = X^T X$$

$$\frac{\partial^2 Q(\bar{W})}{\partial \bar{W}^2} = (X^T X)^T = X^T X$$

This is a PSD matrix

Because $v^T X^T X v = \|Xv\|_2^2 \ge 0$

for any vector v

$\therefore \frac{1}{2} \| \bar{W}[j] X^T - Y^T[j] \|_2^2$ is a convex function of $\bar{w}[j]$

$\therefore Q(\bar{w})$ is a convex function of $\bar{w}$

(c) $$Q(\bar{w}) = \frac{1}{2} \times \left\| \begin{pmatrix} \bar{W}[1] \\ \bar{W}[2] \\ \vdots \\ \bar{W}[p] \end{pmatrix} (X_1\ X_2 \cdots X_n) - (Y_1\ Y_2 \cdots Y_n) \right\|_F^2$$

$$\therefore \frac{\partial Q(\bar{w})}{\partial \bar{W}} = \left(\nabla_{\bar{w}} Q(\bar{w}) \right)^T = \left(\bar{W} X^T X - Y^T X \right)^T$$

$$= X^T X \bar{W}^T - X^T Y$$

(d) $\because \min_{i=1}^{L} d_i = \min\{p, d\}$

$\therefore W_0 = \bar{W}$

$Q(W_0) = \frac{1}{2} \| W_0 X^T - Y^T \|_F^2$

$\because \nabla_{W_0} Q(W_0) = W_0 X^T X - Y^T X$

$= \bar{W} X^T X - Y^T X = 0$

$\because Q(W_0)$ is a convex function of W_0

$\therefore W_0 = \bar{W}$ is a global minimizer of $Q(\bar{W})$ and $\hat{R}(f_{W_{1:L}})$

(e) $$\frac{\partial \hat{R}(f_{W_{1:L}})}{\partial W_i} = \bar{W}_{(L:i+1)}^T \frac{\partial \hat{R}(f_{W_{1:L}})}{\partial \bar{W}} \bar{W}_{(i-1:1)}^T$$

If we cannot guarantee that $\bar{W} = W_0$ then we cannot guarantee the convexity of $\hat{R}(f_{W_{1:L}})$

$\therefore$ The critical point of $\hat{R}$ may not be Global minimum

(f) The sufficient condition can be All $W_1 \ldots W_L$ are of full rank.

3.

(a) The relevant code is in the appendix

(b) The relevant code and graph are in the appendix

$d=10$ takes more iterations to decrease the loss to 0 than $d=5$.

But when $d=30$, The loss will oscillate around 1 and will not go to 0 at the end of 50000 iterations.

(c) (i) $\nabla R(P) = \nabla E_x \{ \max\{0, 1 - f_P(x) \cdot (-1)^{V^Tx}\} \}$

$$= E_x \{ \nabla_P \{ \max\{0, 1 - f_P(x) \cdot (-1)^{V^Tx}\} \} \}$$

$$= E_x \{ \nabla_P [(-1)^{V^Tx} - f_P(x)] 1_1 (f_P(x) \cdot (-1)^{V^Tx}) \}$$

$$= E_x \{ \nabla_P [y - f_P(x)] 1_1 (f_P(x) \cdot y) \}$$

$$= E_{x,y} \{ \nabla_P [y\, 1_1 (f_P(x) \cdot y)] - \nabla_P [f_P(x) \cdot 1_1 (f_P(x) \cdot y)] \}$$

$$= -E_{x,y} [\nabla_P [f_P(x) \cdot 1_1 (f_P(x) \cdot y)]] + E_{x,y} \{ y\, \underbrace{\nabla_P [1_1 (f_P(x) \cdot y)]} \}$$

$$E_x \{ (-1)^{V^Tx} \cdot \underbrace{\nabla_P [1_1 (f_P(x) \cdot (-1)^{V^Tx})]}_{g(x)} \}$$

$$- E_{x,y} [\nabla_P [f_P(x) \cdot 1_1 (f_P(x) \cdot y)]]$$

$$= - E_x [\nabla_P (f_P(x) \cdot 1_1 (f_P(x) \cdot (-1)^{V^Tx}))]$$

$=$ when $(-1)^{V^Tx} = 1$:

$$1_1 (f_P(x) \cdot (-1)^{V^Tx}) = 1_1 (f_P(x))$$

when $(-1)^{V^Tx} = -1$:

$$1_1 (f_P(x) \cdot (-1)^{V^Tx}) = 1_1 (-f_P(x))$$

$$\therefore - E_x [\nabla_P (f_P(x)) \cdot 1_1 (f_P(x) \cdot (-1)^{V^TX})]$$

$$= - E_x \left[\frac{1_1 (f_P(x)) - 1_1 (-f_P(x))}{1 - (-1)} \nabla_P (f_P(x)) \right]$$

$$= \underbrace{- E_x \left[\frac{1_1 (f_P(x)) - 1_1 (-f_P(x))}{2} \cdot \nabla_P (f_P(x)) \right]}_{a}$$

$$\therefore \nabla R(P) = a + E_x [(-1)^{V^TX} g(x)]$$

where $g(x) = \nabla_P [1_1 (f_P(x) \cdot (-1)^{V^Tx})]$

c (iii)

$$g(x) = \nabla_P \, l_1 [f_P(x) \cdot (-1)^{v^T x}]$$

$$E_v [| \nabla R(P) - a |^2] = \underbrace{\frac{1}{2^d}}_{\substack{v \text{ is } (0,1)^d \\ \therefore 2^d \text{ possible } v}} \sum_v \left(E_x \left((-1)^{v^T x} g(x) \right) \right)^2 \le \frac{E_x [g(x)^2]}{2^d}$$

$$g(x) = \nabla_P \, l_1 [f_P(x) \cdot (-1)^{v^T x}]$$

$$= -\frac{1}{2} \nabla_P f_P(x) \cdot [l_1(f_P(x)) + l_1(-f_P(x))]$$

$$g(x)^2 \le \left(\nabla_P f_P(x) \cdot \frac{l_1(f_P(x)) + l_1(-f_P(x))}{2} \right)^2$$

$$\because l_1(f_P(x)) + l_1(-f_P(x)) \le 2$$

$$\therefore g(x)^2 \le (\nabla_P f_P(x))^2 = \sum_j (\nabla_{P_j} f_P(x)_j)^2$$

$$\le d C_P^2$$

$$\therefore E_v [| \nabla R(P) - a |^2] \le \frac{E_x [g(x)^2]}{2^d} \le \frac{d C_P^2}{2^d}. \quad \therefore C_P = d C_P^2$$

(d) As we go deeper, the gradient will be the product of more gradients, which will be more unstable.
And for sigmoid activation function, it has 0 gradient at top and bottom.
As a result, we will highly likely stuck at 0 gradient.
Mainly because there are more gradients time together, easier to get 0 gradient even when d is small.

Appendix

3

(a)

```python
class Net(nn.Module):
    def __init__(self, d):
        super(Net, self).__init__()
        # TO IMPLEMENT Part (a)
        # here you should declare functions for layers with parameters
        self.fc1 = nn.Linear(d, 10*d, bias=True)
        self.fc2 = nn.Linear(10*d, 1, bias=True)

    def forward(self, x):
        # TO IMPLEMENT Part (a)
        # here you should trace the forward computation of the network
        # using functions declared in __init__
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x.float()

```

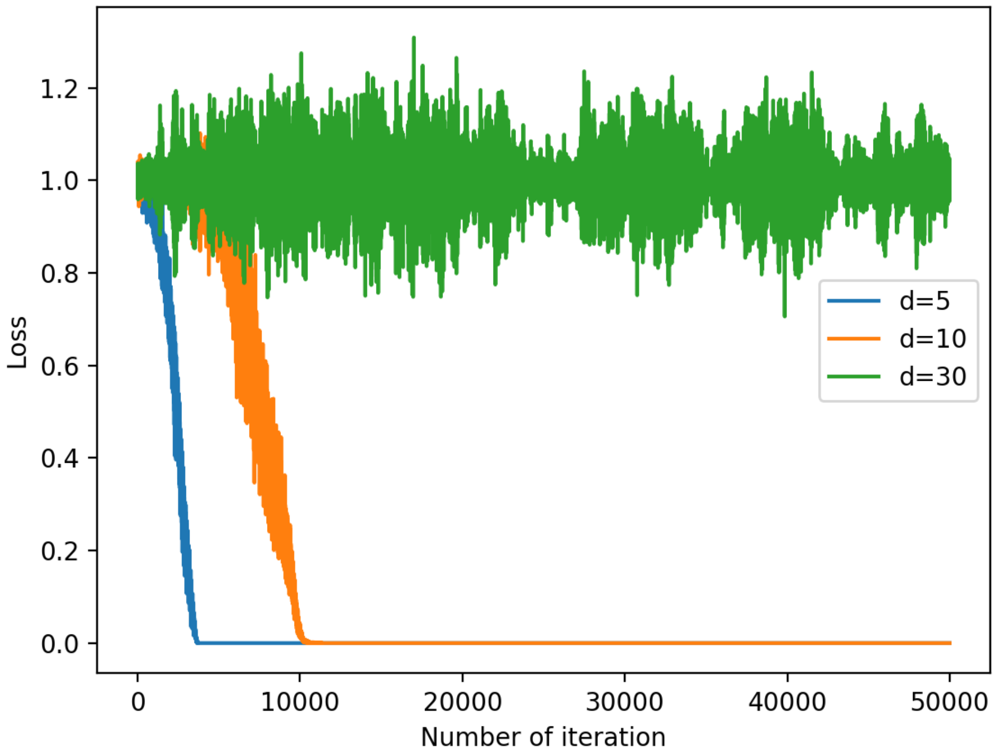
```python
# Initializing the data generation process
d = 5
data = DataGen(d)

### Defining network parameters by hand

# Initializing the network
net = Net(d)

# Defining and setting parameters for perfect approximation
# TO IMPLEMENT Part (a)
v = data.v_star
w = np.zeros((10*d, d))
for i in range(10*d):
    if i <= 3*d/2-1:
        w[i,:] = v
b = np.zeros((10*d, 1))
alpha = np.zeros((1, 10*d))
for j in range((d//2)+1):
    b[3*j,0] = -(2*j-0.5)
    b[3*j+1,0] = -2*j
    b[3*j+2,0] = -(2*j+0.5)
    alpha[0,3*j] = 4
    alpha[0,3*j+1] = -8
    alpha[0,3*j+2] = 4
beta = np.array([-1])
set_parameters(d, w, b, alpha, beta, net)

# Checking that network perfectly approximates 10,000 examples
inputs, labels = data.get_batch(10000)
assert sum(net(inputs).reshape(-1) - labels.float()) == 0
```

(b)

```python
# Defining the loss
def myLoss(outputs, labels):
    # TO IMPLEMENT Part (b)
    # should return 1/b * sum_{i=1}^b loss(f_p(x_i),y_i)
    outputs = outputs.view(-1,1)
    labels = labels.view(-1,1)
    n = outputs.size()[0]
    ones = torch.ones(n, 1, dtype=float, requires_grad=True)
    arg = torch.mul(outputs, labels)
    arg = torch.mul(arg, -1)
    arg = torch.add(arg, 1)
    l = F.relu(arg)
    ml = torch.sum(l)
    ml = torch.div(ml, n)
    return ml
```

```python
d_list = [5, 10, 30]
for d1 in d_list:
    net1 = Net(d1)
    data1 = DataGen(d1)
    loss_function1 = myLoss
    optimizer1 = optim.SGD(net1.parameters(), lr=0.001, momentum=0.9)
    batch_size1 = 100
    num_iter1 = 5*10**4
    loss_over_iteration1 = []
    for m in range(num_iter1):
        inputs1, labels1 = data1.get_batch(batch_size1)
        optimizer1.zero_grad()
        outputs1 = net1(inputs1)
        loss1 = loss_function1(outputs1, labels1)
        loss1.backward()
        optimizer1.step()
        loss_over_iteration1.append(loss1.item())
        if m%200 == 0:
            print('[%5d] loss: %.3f' % (m, loss1.item()))
    print('Finished Training d=%2d' % d1)
    plt.plot(np.array(loss_over_iteration1))
plt.legend(('d=5', 'd=10', 'd=30'))
plt.xlabel('Number of iteration')
plt.ylabel('Loss')
plt.show()
```

Plot



1.2
1.0
0.8
0.6
0.4
0.2
0.0
Loss
0
10000
20000
30000
40000
50000
Number of iteration
d=5
d=10
d=30