Homework 1

Hijun (Jane) Seo
1001423284

Question 1

1.1

$$W^{(1)} = \begin{bmatrix} 2 & 0 & 0 & -2 \\ 0 & 2 & 0 & -2 \\ 0 & 0 & 2 & -2 \end{bmatrix}$$

$$b^{(1)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$w^{(2)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

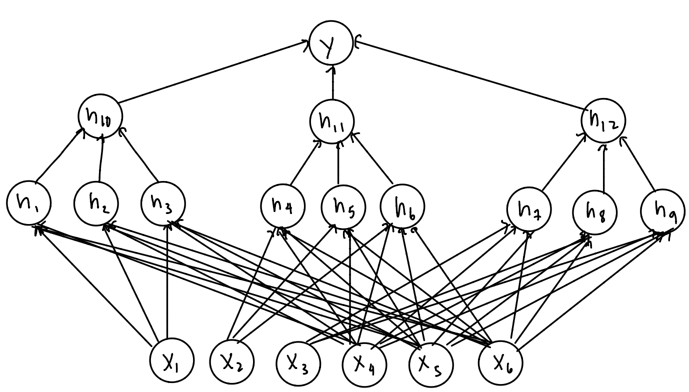
$$b^{(2)} = -2$$

1.2 By combining 3 of the multilayer perceptrons in 1.1, you can check for permutation.

i.e. hidden layers checks if:

x_1 in $\{x_4, x_5, x_6\}$
x_2 in "
x_3 in "

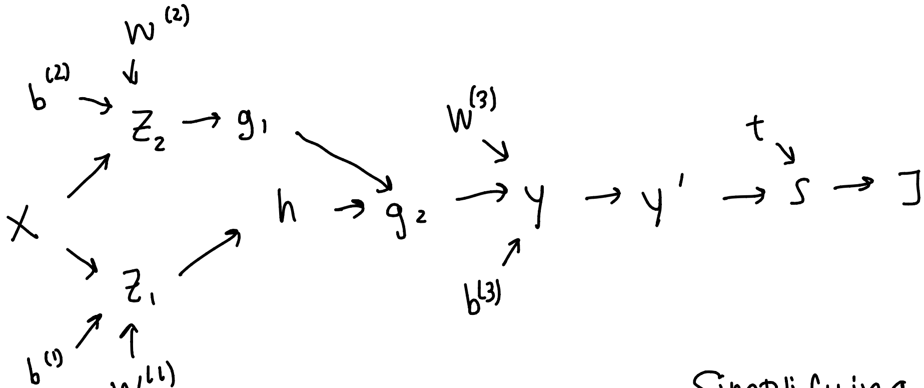
and output layer checks if 3 hidden layers were activated (ie returned a 1)



☆ no arrow means weight = 0

Question 2

2.1.1



Simplifying S:

$$S = \sum_{k=1}^{N} \mathbb{I}(t=k)\log(\mathbf{y}'_k)$$

$$= \sum_{k=1}^{N} t_k \log(\mathbf{y}'_k)$$

$$= \mathbf{t}^T \log \mathbf{y}'$$

$$\bar{J} = 1$$

$$\bar{S} = \bar{J}\frac{dJ}{dS} = -1$$

$$\bar{\mathbf{y}}' = \bar{S}\frac{dS}{d\mathbf{y}'}$$

$$= \bar{S}(\mathbf{t} \oslash \mathbf{y}')$$

element-wise division

$$\bar{\mathbf{y}} = \bar{\mathbf{y}}'\frac{d\mathbf{y}'}{d\mathbf{y}}$$

$$= \bar{\mathbf{y}}'\frac{d(\mathrm{softmax}(\mathbf{y}))}{d\mathbf{y}}$$

$$= \bar{\mathbf{y}}'^T \mathbf{J} \quad \text{where} \quad J = \begin{bmatrix} s(y_1)-s(y_1)^2 & \cdots & 0-s(y_1)s(y_N) \\ \cdots & s(y_j)-s(y_j)s(y_i) & \cdots \\ 0-s(y_N)s(y_1) & \cdots & s(y_N)-s(y_N)^2 \end{bmatrix}$$

*s = softmax

2.1.1 (cont'd)

$$\bar{W}^{(3)} = \bar{y} \frac{dy}{dW^{(3)}}$$

$$= \bar{y} g_2^{T}$$

$$\bar{b}^{(3)} = \bar{y}$$

$$\bar{g}_2 = \bar{y} \frac{dy}{dg_2}$$

$$= W^{(3)T} \bar{y}$$

$$\bar{h} = \bar{g}_2 \frac{dg_2}{dh}$$

$$= \operatorname{diag}(g_2)\, \bar{g}_2$$

$$\bar{g}_1 = \bar{g}_2 \frac{dg_2}{dg_1}$$

$$= \operatorname{diag}(h)\, \bar{g}_2$$

$$\bar{z}_1 = \bar{h} \frac{dh}{dz_1}$$

$$= \begin{cases} 0 & \text{if } z_1 < 0 \\ \bar{h} & \text{if } z_1 > 0 \end{cases}$$

$$\bar{z}_2 = \bar{g}_1 \frac{d\bar{g}_1}{dz_2}$$

$$= \bar{g}_1 \odot \sigma'(z_2)$$

$$= \bar{g}_1 \odot \frac{e^{-z_2}}{(1+e^{-z_2})^2}$$

$$= \bar{g}_1 \odot (\sigma(z_2)(1-\sigma(z_2)))$$

$$\bar{W}^{(2)} = \bar{z}_2 \frac{dz_2}{dW^{(2)}} = \bar{z}_2 x^{T}$$

$$\bar{b}^{(2)} = \bar{z}_2 \frac{dz_2}{db^{(2)}} = \bar{z}_2$$

$$\bar{W}^{(1)} = \bar{z}_1 \frac{dz_1}{dW^{(1)}} = \bar{z}_1 x^{T}$$

$$\therefore \quad \bar{x} = \bar{z}_1 \frac{dz_1}{dx} + \bar{z}_2 \frac{dz_2}{dx} = W^{(1)T} \bar{z}_1 + W^{(2)T} \bar{z}_2$$

2.2.2

$$\mathcal{L}(x) = x^T v v^T x$$

$$\therefore H = 2 v v^T$$

To store input (only v) $= n$

To calculate Hessian (# of scalar mult.) $= \frac{n(n+1)}{2}$ since matrix is symmetrical

To store output (as matrix) $= n^2$

$$\therefore O(\tfrac{3}{2} n^2)$$

2.3 Backpropagation:

(constant 2 was omitted)

$$M = v^T y$$

$$= \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 1 + 2 + 3 = 6$$

$$Z = vM = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} 6 = \begin{bmatrix} 6 \\ 12 \\ 18 \end{bmatrix}$$

$$\therefore Z^T = [6, 12, 18]$$

To store inputs: $2n$

To calculate M: n scalar multiplications

To calculate Z: n scalar multiplications

To store output: n

$$\therefore O(5n)$$

2.3 (cont'd)

Forward-mode:

$$H = vv^T$$

$$= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

$$z = Hy$$

$$= \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 12 \\ 18 \end{bmatrix}$$

$\therefore z^T = [6, 12, 18]$

To store inputs : $2n$

To calculate $H = vv^T$: n^2 scalar multiplications

To calculate z : n^2 scalar multiplications

To store output : n

$\therefore O(2n^2)$

2.4 $Z = Hy_1y_2^T = vv^Ty_1y_2^T$

(constant 2 is omitted as per Q2.3)

Input & output memory costs are the same for both methods.

Input $= 2n+m$ Output $= nm$

Reverse-mode:

$\therefore 5mn+2n+2m$

$$Z = vv^T \underbrace{y_1y_2^T}_{=a} = v\underbrace{v^Ta}_{=b} = \underbrace{vb}$$

a: memory: nm, Scalar mult: nm

b: memory: m, Scalar mult: nm

vb: Scalar mult: nm

2.4 (cont'd)

Forward-mode:

$$Z = \underbrace{VV^T}_{= a} y_1 y_2^T = \underbrace{a y_1}_{= b} y_2^T = \underbrace{b y_2^T}_{\text{Scalar mult : } nm}$$

$\therefore 3n^2 + 2nm + 3n + m$

memory : n^2
Scalar mult : n^2

memory : n
Scalar mult : n^2

Looking at the highest order terms

Reverse : $5mn$ Forward : $3n^2 + 2nm$

$\therefore$ if $m > n$, forward-mode is better

Question 3

3.2 Gradient of loss $= \frac{2}{n} X^T (X\hat{w} - t) = 0$

$$X^T X \hat{w} - X^T t = 0$$

$$X^T X \hat{w} = X^T t$$

Since $X^T X$ is invertible when $n > d$,

$$\therefore \hat{w} = (X^T X)^{-1} X^T t$$

3.3.2 $\quad w_0 = 0 \quad$ let $\alpha = \frac{n}{2}$ (to make calculation simpler)

Iteration 1: $\quad w_1 \leftarrow w_0 - \frac{2\alpha}{n} X^T (X w_0 - t)$

$$w_1 \leftarrow X^T t$$

Iteration 2: $\quad w_2 \leftarrow w_1 - \frac{2\alpha}{n} X^T (X w_1 - t)$

$$w_2 \leftarrow X^T t - X^T (X X^T t - t)$$

$$w_2 \leftarrow X^T t - X^T X X^T t + X^T t$$

$$w_2 \leftarrow 2 X^T t - X^T X X^T t$$

$$w_2 \leftarrow X^T (2 \mathbb{I} - X^T X X^T) t$$

Iteration 3: $\quad w_3 \leftarrow w_2 - \frac{2\alpha}{n} X^T (X w_2 - t)$

$$w_3 \leftarrow 2 X^T t - X^T X X^T t - X^T X (2 X^T t - X^T X X^T t) + X^T t$$

$$\leftarrow 3 X^T t - X^T X X^T t - 2 X^T X X^T t + X^T X X^T X X^T t$$

$$\leftarrow 3 X^T t - 3 X^T X X^T t + X^T X X^T X X^T t$$

$$\leftarrow X^T [3 \mathbb{I} - 3 X X^T + X X^T X X^T] t$$

$\therefore$ the pattern is with every iteration only the term in the brackets changes

$$\therefore \quad w_\infty = X^T A t$$

3.3.2 (cont'd)

Solving for A:

$$\text{Gradient of loss} = \frac{2}{n} X^T (X\hat{w} - t) = 0$$

If $\hat{w} = X^T A t$,

$$X X^T A t - t = 0$$

$$(X X^T A - I) t = 0$$

$$X X^T A = I$$

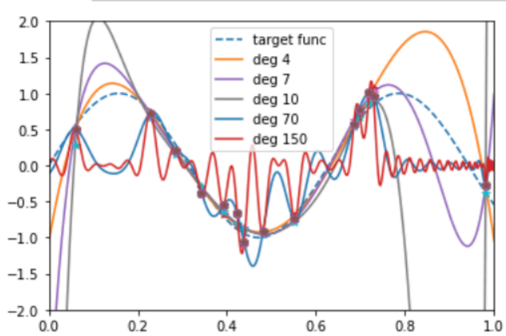
Since XX^T is invertible if $d > n$ $\therefore$ $A = (XX^T)^{-1}$

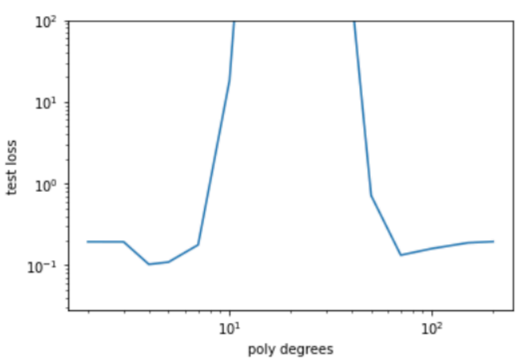
$\therefore$ $\hat{w} = X^T (XX^T)^{-1} t$

3.3.3

In [8]:
```
# to be implemented; fill in the derived solution for the underparameterized (d<n) and overparameterized (d>n) problem

def fit_poly(X, d,t):
  X_expand = poly_expand(X, d=d, poly_type = poly_type)
  n = X.shape[0]
  if d > n:
    ## W = ... (Your solution for Part 3.3.2)
    W = X_expand.T @ np.linalg.inv(X_expand @ X_expand.T) @ t
  else:
    ## W = ... (Your solution for Part 3.2)
    W = np.linalg.inv(X_expand.T @ X_expand) @ X_expand.T @ t
  return W
```





This plot shows that the losses with lower degree polynomials (< 7) are similar to the losses with higher degree polynomials (> 70).
$\therefore$ no, overparameterization doesn't always lead to overfitting.