Guolun Li

1.1

$$h_i = \phi(2(x_4 - x_i)) = \begin{cases} 1, & \text{if } x_4 = x_i \\ 0, & \text{otherwise} \end{cases}$$

$$y = \phi(2(1 - \Sigma h_i)) = \begin{cases} 1, & \text{if } h_i = 1 \text{ for some } i \in \{1,2,3\} \\ 0, & \text{otherwise} \end{cases}$$

Thus, $w^{(1)}\vec{x} + b^{(1)} = 2\begin{pmatrix} x_4 - x_1 \\ x_4 - x_2 \\ x_4 - x_3 \end{pmatrix}$

$$\Rightarrow \boxed{w^{(1)} = 2\begin{pmatrix} -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix}}, \boxed{b^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}}$$

and $w^{(2)}\vec{h} + b^{(2)} = 2(1 - \Sigma h_i)$

$$\Rightarrow \boxed{w^{(2)} = -2\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}, \quad b^{(2)} = -2}$$

1.2 We use a brute force approach (list out all permutations) to achieve the goal.

We'll construct two hidden layers and one output layer.

The first hidden layer compares, one by one, the first three nodes to the last three nodes. The second hidden layer enumerates all possibilites that the last three nodes are a permutation of the first three nodes.

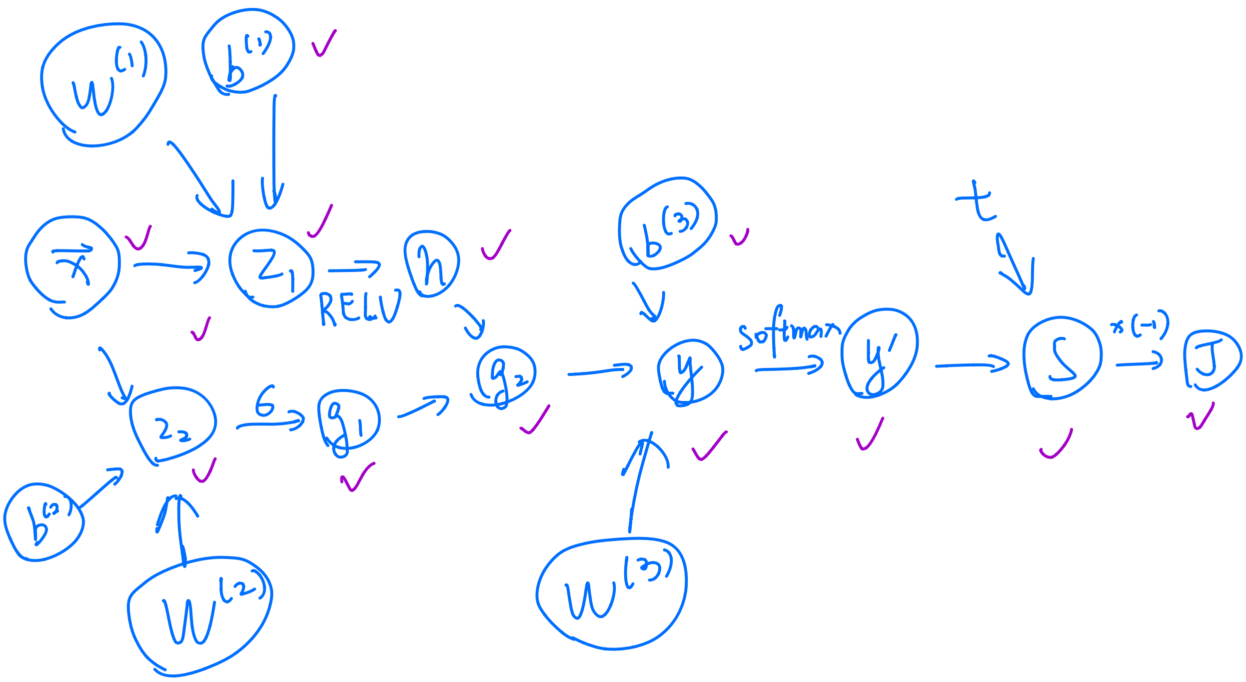
The first hidden layer has 9 nodes, each node checks if $x_i = x_j$, for some $i \in \{1,2,3\}$, $j \in \{4,5,6\}$. This is similar to Q1.1, and the activation function would be $\phi_1(z) = I(z \in [-1,1])$.

The second hidden layer has 6 nodes, each node checks if $(x_1, x_2, x_3) = (x_i, x_j, x_k)$, where (x_i, x_j, x_k) is a permutation of (x_4, x_5, x_6). This can be done by putting weight 1 on the nodes that check $x_1 = x_i$, $x_2 = x_j$, $x_3 = x_k$, and weight 0 on other nodes, then using an activation function $\phi_2(z) = I(z = 3)$.

Then, the output layer checks if any node in the second hidden layer is 1. This is also similar to Q1.1, we simply put weight 1 on each node and use $\phi(z) = I(z = 1)$.

2.1.1 Computational Graph [0pt]

Draw the computation graph relating $\mathbf{x}$, t, $\mathbf{z}_1$, $\mathbf{h}$, $\mathbf{z}_2$, $\mathbf{g}_1$, $\mathbf{g}_2$, $\mathbf{y}$, $\mathbf{y}'$, $\mathcal{S}$ and $\mathcal{J}$.



2.1.2 Backward Pass [1pt]

Derive the backprop equations for computing $\bar{\mathbf{x}} = \frac{\partial \mathcal{J}}{\partial \mathbf{x}}$, one variable at a time, similar to the vectorized backward pass derived in Lec 2.

$\bar{J} = 1$

$\bar{S} = -\bar{J}$

$\bar{y}' = \bar{S}\,\nabla_{y'} S = -\bar{J} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \frac{1}{y'_t} \\ \vdots \\ 0 \end{pmatrix} \leftarrow t^{th}$ position , since $\frac{\partial S}{\partial y_k} = \frac{\partial}{\partial y_k}(\log y'_t) = \begin{cases} \frac{1}{y'_t}, & \text{if } k=t \\ 0, & \text{o.w.} \end{cases}$

$$\overline{y} = \left(\frac{\partial y'}{\partial y}\right)^T \overline{y'}, \quad \text{where} \left(\frac{\partial y'}{\partial y}\right)_{i,j} = I(i=j)\,\mathrm{softmax}(y_i) - \mathrm{softmax}(y_i)\cdot\mathrm{softmax}(y_j)$$

(see lemma 1)

$$\overline{g_2} = (W^{(3)})^T \overline{y},$$

$$\overline{h} = \overline{g_2} \odot g_1,$$

$$\overline{g_1} = \overline{g_2} \odot h,$$

$$\overline{z_2} = \sigma'(z_2) \odot \overline{g_1}, \quad \text{where } \sigma'(x) = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$\overline{z_1} = \mathrm{RELU}'(z_1) \odot \overline{h}, \quad \text{where } \mathrm{RELU}'(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

(Define $\mathrm{RELU}'(0) = 1$ for computation purpose)

$$\overline{x} = W^{(1)T}\overline{z_1} + W^{(2)T}\overline{z_2}$$

Lemma 1:

$$\left(\frac{\partial y'}{\partial y}\right)_{i,j} = \frac{\partial y'_i}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\frac{e^{y_i}}{\sum e^{y_k}}\right)$$

$$= \frac{\partial_{y_j} e^{y_i} \sum e_k - e^{y_j}\cdot e^{y_i}}{(\sum e^{y_k})^2}$$

$$= \frac{1}{(\sum e^{y_k})^2}\left[I(i=j)\, e^{y_i} \cdot \sum e^{y_k} - e^{y_i + y_j}\right]$$

$$= I(i=j)\,\mathrm{softmax}(y_i) - \mathrm{softmax}(y_i)\cdot\mathrm{softmax}(y_j)$$

2.2.2 Computation Cost [1pt]

What is the number of scalar multiplications and memory cost of computing the Hessian $\mathbf{H}$ in terms of n?

2.2.2

$L = x^T V V^T x$

$\nabla L = V V^T x$

$H = \nabla^2 L = 2 V V^T$ which requires $2n^2$ multiplications and

$\Downarrow$

$O(n^2)$

$n^2 + n$ memory cost

$\Downarrow$

$O(n^2)$

v v y
n×1 1×n n×1

2.3 Vector-Hessian Products [1pt]

Compute $\mathbf{z} = \mathbf{H}\mathbf{y} = \mathbf{v}\mathbf{v}^\top\mathbf{y}$ where $n = 3$, $\mathbf{v}^\top = [1, 2, 3]$, $\mathbf{y}^\top = [1, 1, 1]$ using two algorithms: reverse-mode and forward-mode autodiff.

In backpropagation (also known as reverse-mode autodiff), you will compute $\mathbf{M} = \mathbf{v}^\top\mathbf{y}$ first, then compute $\mathbf{v}\mathbf{M}$. Whereas, in forward-mode, you will compute $\mathbf{H} = \mathbf{v}\mathbf{v}^\top$ then compute $\mathbf{H}\mathbf{y}$.

Write down the numerical values of $\mathbf{z}^T = [z_1, z_2, z_3]$ for the given $\mathbf{v}$ and $\mathbf{y}$. What is the time and memory cost of evaluating $\mathbf{z}$ with backpropagation (reverse-mode) in terms of n? What about forward-mode?

For Q 2.3 and Q 2.4, we use the lemma that computing the matrix product MN, where M is $a \times b$, N is $b \times c$, takes in total abc scalar multiplications. This is easy to see, as each entry of MN is the dot product of one row from M and one column from N (takes b steps), and there are ac such entries.

$z^T = (6 \quad 12 \quad 18)$

Back Prop:

Time: $M = v^T y$, $z = vM$

Total time $= n + n = 2n$

Memory: y, v, M, z

Total Memory $= n + n + 1 + n = 3n+1$

Forward:

Time: $H = vv^T$, $z = Hy$

Total time $= n^2 + n^2 = 2n^2$

Memory: v, H, y, z

Total Memory $= n + n^2 + n + n = n^2 + 3n$

$n{\times}1{\times}1{\times}n{\times}n{\times}1{\times}1{\times}m$

2.4 Trade-off of Reverse- and Forward-mode Autodiff [1pt] $Z = vv^T y_1 y_2^T$

$H = vv^T$

Consider computing $\mathbf{Z} = \mathbf{H}\mathbf{y}_1\mathbf{y}_2^\top$ where $\mathbf{v} \in \mathbb{R}^{n\times 1}, \mathbf{y}_1 \in \mathbb{R}^{n\times 1}$ and $\mathbf{y}_2 \in \mathbb{R}^{m\times 1}$. What are the time and memory cost of evaluating $\mathbf{Z}$ with reverse-mode in terms of n and m? What about forward-mode? When is forward-mode a better choice? (Hint: Think about the shape of Z, "tall" v.s. "wide".)

Back Prop:

Time: $B_1 = y_1 y_2^T$, $B_2 = v^T B_1$, $Z = v B_2$

Total time $= nm + nm + nm = 3mn$

Memory: y_1, y_2, B_1, v, B_2, Z

Total Memory $= n + m + nm + n + m + nm = 2(m+n+mn)$

Forward:

Time: $A_1 = vv^T$, $A_2 = A_1 y_1$, $Z = A_2 y_2^T$

Total time $= n^2 + n^2 + nm = n(2n+m)$

Memory: v, A_1, y_1, A_2, y_2, Z

Total Memory $= n + n^2 + n + n + m + mn = 3n + m + n^2 + mn$

$\text{Back Prop}_{time} - \text{Forward}_{time} = n(2m-2n)$

$\text{Back Prop}_{Memory} - \text{Forward}_{Memory} = (n+1)(m-n)$

Thus, given the dimension of Z being $n \times m$, using back propagation is better when $m < n$, using forward mode is better when $m > n$.