CSDS 440: Assignment 5

Shaochen (Henry) ZHONG, sxz517

Mingyang TIE, mxt497

Due on 10/09/2020, submitted early on 10/02/2020

Fall 2020, Dr. Ray

Problem 19

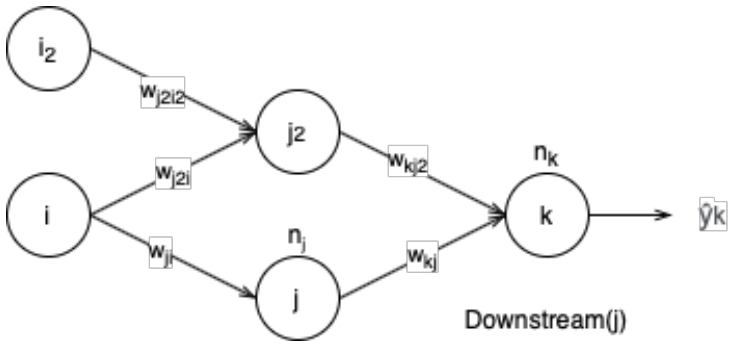
$$
\begin{aligned}
b^T u &= u^T b \\
&\leq u^T \cdot Ax \ \text{ (Since } Ax \geq b\text{)} \\
c^T X &= x^T c \\
&\geq x^T A^T u \ \text{ (Since } A^T u \leq c\text{)} \\
&\geq u^T Ax \\
\Rightarrow\ & b^T u \leq u^T Ax \leq c^T X \\
\Longrightarrow\ & b^T u \leq c^T X
\end{aligned}
$$

The statement is therefore proven.

Problem 20

We denote $h(u)$ to be the activation function for u and we do not make assumption of which activation function is used (so we won't have the sigmoid simplication of $\frac{\partial h}{\partial u} = h(u)(1 - h(u))$). For traning example α, $h(\alpha) = o_\alpha$ (representing the prediction output), and we denotes $y = y_\alpha$ for the output label.

For the ease of understanding, we denote our symbols with respect to the following NN, which use almost identical notation to the one taught in *Lecture 10*.



Hidden-to-Output

$$
\begin{aligned}
\frac{\partial L}{\partial w_{ji}} &= \frac{\partial L}{\partial n_j}\frac{\partial n_j}{w_{ji}} \\
&= \frac{\partial L}{\partial n_j} \cdot \frac{\partial (w_{ji} \cdot x_{ji})}{w_{ji}} \\
&= \frac{\partial L}{\partial n_j} \cdot x_{ji} \\
&= \frac{\partial L}{\partial h(n_j)}\frac{\partial h(n_j)}{\partial n_j} \cdot x_{ji} \\
&= \frac{\partial}{\partial h(n_j)}(-y\log(h(n_j)) - (1-y)\log(1-h(n_j))) \cdot \frac{\partial h(n_j)}{\partial n_j} \cdot x_{ji} \\
&= (-\frac{-y}{h(n_j)} + \frac{1-y}{1-h(n_j)}) \cdot \frac{\partial h(n_j)}{\partial n_j} \cdot x_{ji} \\
&= (\frac{1-y}{1-h(n_j)} - \frac{-y}{h(n_j)}) \cdot \frac{\partial h(n_j)}{\partial n_j} \cdot x_{ji}
\end{aligned}
$$

Base on the calucation of the output layer, we may tall the backpropagation weight updates for hidden-to-output layer is:

$$
\frac{\partial L}{\partial w_{ji}} = (\frac{1-y}{1-h(n_j)} - \frac{-y}{h(n_j)}) \cdot \frac{\partial h(n_j)}{\partial n_j} \cdot x_{ji}
$$

Input-to-Hidden

Since j affects the $\hat{y}$ outputs only through $Downstream(j)$, we have:

$$\begin{aligned}
\frac{\partial L}{\partial w_{ji}} &= \frac{\partial L}{\partial n_j} \cdot \frac{\partial n_j}{\partial w_{ji}} \\
&= \frac{\partial L}{\partial n_j} \cdot \frac{\partial (w_{ji} \cdot x_{ji})}{w_{ji}} \\
&= \frac{\partial L}{\partial n_j} \cdot x_{ji} \\
&= \sum_{k \in Downstream(j)} \frac{\partial L}{\partial n_k} \cdot \frac{\partial n_k}{\partial n_j} \cdot x_{ji} \\
&= \sum_{k \in Downstream(j)} \frac{\partial L}{\partial n_k} \cdot \frac{\partial (w_{kj} h(n_j)}{\partial n_j} \cdot x_{ji} \\
&= \sum_{k \in Downstream(j)} \frac{\partial L}{\partial n_k} \cdot w_{kj} \frac{\partial h(n_j)}{\partial n_j} \cdot x_{ji} \\
&= \frac{\partial h(n_j)}{\partial n_j} \cdot \sum_{k \in Downstream(j)} \frac{\partial L}{\partial n_k} \cdot w_{kj} \cdot x_{ji} \\
&= \frac{\partial h(n_j)}{\partial n_j} x_{ji} \cdot \sum_{k \in Downstream(j)} \frac{\partial L}{\partial w_{kj}} \frac{w_{kj}}{x_{kj}}
\end{aligned}$$

Base on the calucation of the hidden layer, we may tall the backpropagation weight updates for input-to-output layer is:

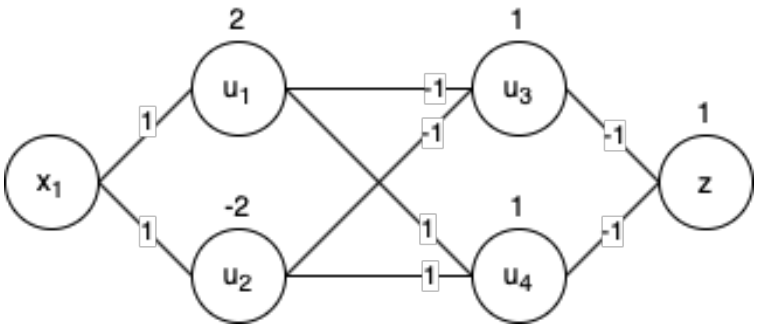
$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial h(n_j)}{\partial n_j} x_{ji} \cdot \sum_{k \in Downstream(j)} \frac{\partial L}{\partial w_{kj}} \frac{w_{kj}}{x_{kj}}$$

I don't know if it is required (as it is now showed on the slide), but if you'd expect us to show an input-to-hidden equation with loss function plugged in, it will be:

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial h(n_j)}{\partial n_j} x_{ji} \cdot \sum_{k \in Downstream(j)} \left(\frac{1-y}{1-h(n_k)} - \frac{-y}{h(n_k)}\right) \cdot \frac{\partial h(n_k)}{\partial n_k} \cdot x_{kj} \cdot \frac{w_{kj}}{x_{kj}}$$

Problem 21

Note for the following neural network, the weights are noted on the edges and the activation thresholds are noted on the top of each neuron (as for the neuron with threshold x will output $+1$ if the input $\geq x$, and output -1 otherwise). Note we omitted x_2 as it is simply a duplication of x_1, and we can design a network without using the former.



Now to trace each example. Note for each neuron, the inequality is the input against the activation threshold and the output is the one in bracket.

x_1	x_2	u_1	u_2	u_3	u_4	z
-4	-4	$-4 < 2$ (-1)	$-4 < -2$ (-1)	$1+1 > 1$ (1)	$-1-1 < 1$ (-1)	$-1+1 < 1$ (-1)
-1	-1	$-1 < 2$ (-1)	$-1 > -2$ (1)	$1-1 < 1$ (-1)	$1-1 < 1$ (-1)	$1+1 > 1$ (1)
1	1	$1 < 2$ (-1)	$1 > -2$ (1)	$1-1 < 1$ (-1)	$1-1 < 1$ (-1)	$1+1 > 1$ (1)
4	4	$4 > 2$ (1)	$4 > -2$ (1)	$1-1 < 1$ (-1)	$1+1 > 1$ (1)	$1-1 < 1$ (-1)

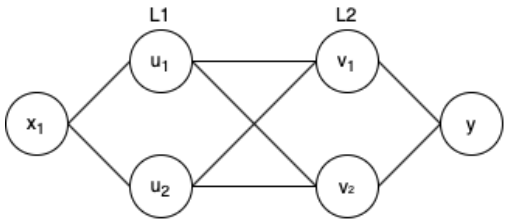
We have showed the network is able to produce the correct classification.

Problem 22

I asked TA, couple of my classmates, and consulted online resources. It seems there are two ways to answer this question. One is to calculate the gradient between each layer and show that without the $sign()$ function the weight will stop updating, but it does not utilized the hint of decision boundary.

So a conceptual argument can be formed on the idea of if we drop the $sign()$ function, a neuron's activation function will simply be its input – which is a linear function of all pervious neurons. This suggests the output of the ANN will also be a linear function of the input, as combination of linear function is always linear. Thus the ANN will have a linear decision boundary, which voids the purpose of having a deep network (as we can rewrite the whole network with just a single layer), and such network won't be useful for solving a problem which requires a non-linear decision boundary.

Below is the gradient calculation approach.



Assume we have the above ANN of two hidden layers, each with m number of nerons, a common square loss will be:

$$\begin{aligned} L(w) &= \frac{1}{2}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \\ &= \frac{1}{2}\sum_{i=1}^{m}(y_i - sign(w \cdot x_i))^2 \end{aligned}$$

For the sake having something differentiable, we drop the $sign()$ function and make it as:

$$L(w) = \frac{1}{2}\sum_{i=1}^{m}(y_i - w \cdot x_i)^2$$

Therefore, for the gradient of L_2 and output, we have:

$$\begin{aligned}\frac{\partial L_2}{\partial w} &= \sum_i^m (y_i - w \cdot x_i)(-x_i) \\ &= \sum_i^m (w \cdot x_i^2 - x_i \cdot y_i)\end{aligned}$$

We furter differentiate it to get the gradient of L_1 and L_2:

$$\begin{aligned}\frac{\partial L_1}{\partial w} &= \frac{\partial}{\partial w}(\sum_i^m (w \cdot x_i^2 - x_i \cdot y_i)) \\ &= \sum_i^m x_i^2\end{aligned}$$

Similiarily, for gradient of input and L_1

$$\frac{\partial L_{\text{input}}}{\partial w} = \frac{\partial}{\partial w} \sum_i^m x_i^2 = 0$$

Since the gradient is 0, the weight of the input will not be updated. Note this is just a 2-layer ANN, an arbitrary ANN with more layers but without the $sign()$ function might also encounter this problem and make the weight updating mechanism ineffective.