Neural Nets and Deep Learning
Assignment 1
COMS 4776: Fall 2025

Aksel Kretsinger-Walters, adk2164
adk2164@columbia.edu
Due: 10/06/2025 2pm 11:59 PM ET

---

(1) Written Problems

(1.1) Hard-Coding Networks

So, for this problem, we need to create a neural network that can identify a superlative superincreasing sequence. Our input vector is $\mathbb{R}^{1\times 4}$, our hidden layer is $\mathbb{R}^{1\times 3}$, and our output layer is $\{0,1\} \in \mathbb{R}^{1\times 1}$. It's recommended to use the hard threshold activation function and the indicator activation function.

My approach to this problem is to use each hidden neuron to test each step of the superincreasing sequence. So the first neuron will test if $x_2 > x_1$, the second neuron will test if $x_3 > x_1 + x_2$, and the third neuron will test if $x_4 > x_1 + x_2 + x_3$. Because superincreasing sequences must be **strictly increasing**, ties will be treated as failures. The output neuron will then use the indicator function to check that all three hidden neurons "passed" their tests.

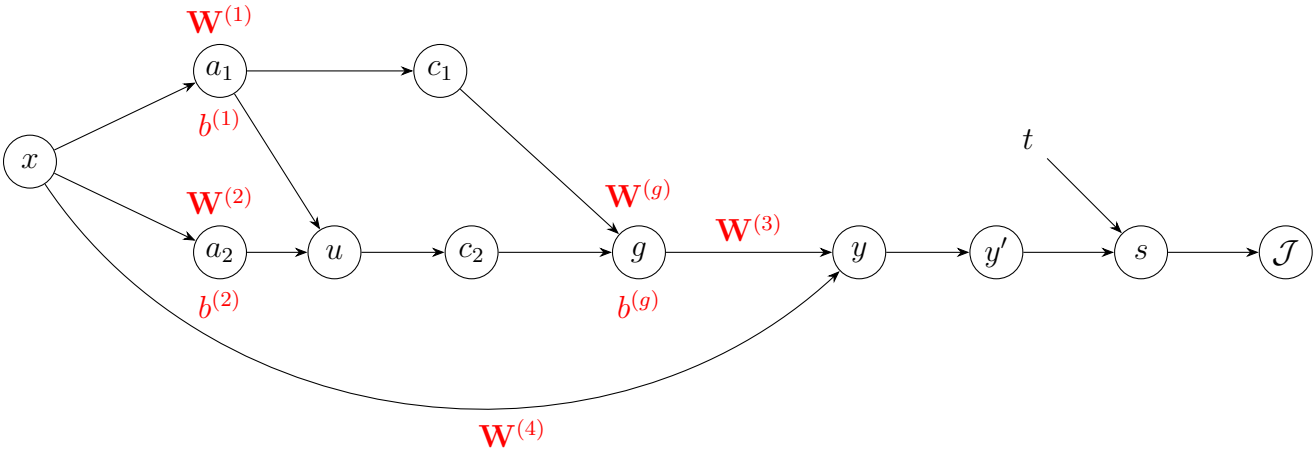
Our hidden layer weights and biases are:

$$\mathbf{W}^{(1)} = \begin{pmatrix} 1 & -1 & 0 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & 1 & 1 & -1 \end{pmatrix} \quad \mathbf{b}^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \mathbf{w}^{(2)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad b^{(2)} = 1$$

Our first hidden neurons are all using the hard threshold activation function, and our output neuron is using the indicator activation function. From row 1, a value of 0 is considered a pass, and a value of 1 is considered a fail. From row 2, any value outside of $[-1, 1]$ is a failure. Hence, the weights of second row are all 1s, and the bias is 1.

(1.2) Backpropagation

Computational Graph



Backwards Pass

$$
\begin{aligned}
\overline{\mathcal{J}} &= 1 \\
\overline{\mathbf{s}} &= -\overline{\mathcal{J}} = -1 \cdot 1 = -1 \\
\overline{\mathbf{y}'} &= \overline{\mathbf{s}}(\mathbf{y} - \mathbf{t}) = \mathbf{t} - \mathbf{y} \\
\overline{\mathbf{y}} &= \overline{\mathbf{y}'} \cdot \mathrm{softmax}'(\mathbf{y}) \\
\overline{\mathbf{g}} &= \mathbf{W}^{(3)T} \cdot \overline{\mathbf{y}} \\
\overline{\mathbf{W}^{(3)}} &= \overline{\mathbf{y}} \cdot \mathbf{g}^T \\
\overline{\mathbf{W}^{(g)}} &= \overline{\mathbf{g}} \cdot \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{pmatrix}^T \\
\overline{b^{(g)}} &= \overline{\mathbf{g}} \\
\overline{c_1} &= \mathbf{W}^{(g)}_{c_1} \cdot \overline{\mathbf{g}} \\
\overline{c_2} &= \mathbf{W}^{(g)}_{c_2} \cdot \overline{\mathbf{g}} \\
\overline{\mathbf{u}} &= \overline{c_2} \cdot \mathrm{ReLU}'(c_2) \\
\overline{\mathbf{W}^{(2)}} &= \overline{\mathbf{u}} \cdot \mathbf{x}^T \\
\overline{b^{(2)}} &= \overline{\mathbf{a}_2} \\
\overline{\mathbf{a}_2} &= \overline{\mathbf{u}} \\
\overline{\mathbf{W}^{(1)}} &= \overline{\mathbf{a}_1} \cdot \mathbf{x}^T \\
\overline{b^{(1)}} &= \overline{\mathbf{a}_1} \\
\overline{\mathbf{a}_1} &= \overline{\mathbf{u}} + \overline{c_1} \cdot \sigma'(a_1) \\
\overline{\mathbf{x}} &= \mathbf{W}^{(1)T} \cdot \overline{\mathbf{a}_1} + \mathbf{W}^{(2)T} \cdot \overline{\mathbf{a}_2} + \mathbf{W}^{(4)T} \cdot \overline{\mathbf{y}}
\end{aligned}
$$

(1.3) Automatic Differentiation

Compute Hessian

Our loss function is

$$L(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x}$$

A quick simplification (note that $\mathbf{v}^T\mathbf{x}$ is a scalar, so it's equal to its own transpose):

$$\begin{aligned} L(\mathbf{x}) &= \frac{1}{2}\mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x} \\ &= \frac{1}{2}(\mathbf{v}^T\mathbf{x})(\mathbf{x}^T\mathbf{v}) \\ &= \frac{1}{2}(\mathbf{v}^T\mathbf{x})^2 \end{aligned}$$

We can see that this is a quadratic form ($\frac{1}{2}\mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x} = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}$).

So, the gradient is:

$$\begin{aligned} \nabla L(\mathbf{x}) &= \frac{1}{2}(\mathbf{A} + \mathbf{A}^T)\mathbf{x} \\ &= \frac{1}{2}(\mathbf{v}\mathbf{v}^T + \mathbf{v}^T\mathbf{v})\mathbf{x} \\ &= \mathbf{v}\mathbf{v}^T\mathbf{x} \end{aligned}$$

And the Hessian is:

$$\begin{aligned} \nabla^2 L(\mathbf{x}) &= \mathbf{A} \\ &= \mathbf{v}\mathbf{v}^T \end{aligned}$$

Now that we have the correct Hessian formula, lets plug in our values for $\mathbf{v}$ and then solve for $\mathbf{x}$

$$\mathbf{v}^T = [3, 1, 4] \implies \mathbf{v} = \begin{pmatrix} 3 \\ 1 \\ 4 \end{pmatrix}$$

$$\begin{aligned} \nabla^2 L(\mathbf{x}) &= \mathbf{v}\mathbf{v}^T \\ &= \begin{pmatrix} 3 \\ 1 \\ 4 \end{pmatrix} \begin{pmatrix} 3 & 1 & 4 \end{pmatrix} \\ &= \begin{pmatrix} 9 & 3 & 12 \\ 3 & 1 & 4 \\ 12 & 4 & 16 \end{pmatrix} \end{aligned}$$

$$\mathbf{x}^T = [2,1,3] \implies \mathbf{x} = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}$$

$$\begin{aligned} \nabla^2 L(\mathbf{x}) &= \begin{pmatrix} 9 & 3 & 12 \\ 3 & 1 & 4 \\ 12 & 4 & 16 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} \\ &= \begin{pmatrix} 9 \cdot 2 + 3 \cdot 1 + 12 \cdot 3 \\ 3 \cdot 2 + 1 \cdot 1 + 4 \cdot 3 \\ 12 \cdot 2 + 4 \cdot 1 + 16 \cdot 3 \end{pmatrix} \\ &= \begin{pmatrix} 57 \\ 19 \\ 76 \end{pmatrix} \end{aligned}$$

Computation Cost The cost of explicitly computing the Hessian using automatic differentiation is $O(n^2)$, where n is the dimension of $\mathbf{x}$. This is because the Hessian is an $n \times n$ matrix, and each entry requires $O(1)$ time to compute. Therefore, the total time complexity is $O(n^2)$.

However, we could compute the Hessian-vector product efficiently using the formula $\nabla^2 L(\mathbf{x}) = \mathbf{v}(\mathbf{v}^T\mathbf{x})$. This would only require $O(n)$ time, since we only need to compute the dot product $\mathbf{v}^T\mathbf{x}$ and then scale $\mathbf{v}$ by that scalar.

1.3.1 Vector-Hessian Products

Hah, looks like I got a bit ahead of myself in the previous section. We'll explicitly compute the vector-Hessian product here.

$$\begin{aligned} \mathbf{z} = \mathbf{H}\mathbf{y} &= (\mathbf{v}\mathbf{v}^T)\mathbf{y} \\ \mathbf{v}^T &= [4,2,1] \\ \mathbf{y}^T &= [2,2,1] \end{aligned}$$

Reverse-mode autodiff:

$$\begin{aligned} \mathbf{v}^T\mathbf{y} &= 4 \cdot 2 + 2 \cdot 2 + 1 \cdot 1 = 13 \\ \mathbf{z} &= 13 \cdot \begin{pmatrix} 4 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 52 \\ 26 \\ 13 \end{pmatrix} \end{aligned}$$

Forward-mode autodiff:

$$\mathbf{v}\mathbf{v}^T = \begin{pmatrix} 16 & 8 & 4 \\ 8 & 4 & 2 \\ 4 & 2 & 1 \end{pmatrix}$$

$$\mathbf{z} = \begin{pmatrix} 16 & 8 & 4 \\ 8 & 4 & 2 \\ 4 & 2 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 52 \\ 26 \\ 13 \end{pmatrix}$$

Computation Efficiency

$$\mathbf{H} = \mathbf{v}\mathbf{v}^T$$
$$\mathbf{Z} = \mathbf{H}\mathbf{y}_1\mathbf{y}_2^T = \mathbf{v}(\mathbf{v}^T\mathbf{y}_1)\mathbf{y}_2^T$$

Let's evaluate how many operations this takes using forward-mode and reverse-mode autodiff.

$$\mathbf{v} \in \mathbb{R}^n, \mathbf{y}_1 \in \mathbb{R}^n, \mathbf{y}_2 \in \mathbb{R}^m$$

Reverse-mode autodiff:

$$\begin{aligned}
\mathbf{y}_2\mathbf{y}_1^T = \mathbf{Y} \in \mathbb{R}^{n\times m} &= O(nm) \text{ operations to compute} \\
\mathbf{v}^T\mathbf{Y} = \mathbf{a} \in \mathbb{R}^{1\times m} &= O(n^2) \text{ operations to compute} \\
\mathbf{v}\mathbf{a} = \mathbf{Z} \in \mathbb{R}^{n\times m} &= O(nm) \text{ operations to compute} \\
\text{Total: } O(nm) + O(n^2) + O(nm) &= O(n^2 + nm) \text{ operations} \\
&= O(n^2) \text{ if } m \leq n. \quad O(nm) \text{ if } m > n
\end{aligned}$$

Forward-mode

$$\begin{aligned}
\mathbf{v}\mathbf{v}^T = \mathbf{H} \in \mathbb{R}^{n\times n} &= O(n^2) \text{ operations to compute} \\
\mathbf{H}\mathbf{y}_1 = \mathbf{z} \in \mathbb{R}^{n\times 1} &= O(n^2) \text{ operations to compute} \\
\mathbf{z}\mathbf{y}_2^T = \mathbf{Z} \in \mathbb{R}^{n\times m} &= O(nm) \text{ operations to compute} \\
\text{Total: } O(n^2) + O(n^2) + O(nm) &= O(n^2 + nm) \text{ operations}
\end{aligned}$$

We can see that if $m < n$ reverse-mode is more efficient, and if $m > n$ forward-mode is more efficient.