Assignment 1

Deadline: March 3, 2025 by 10am.
Assignment structure: There are two sections of problems. One includes written problems, which are presented in this file. The other set involves programming. More information about these are in the Notebook "hw1_code.ipynb".
Submission:

- You must submit your solutions of written problems as a PDF file through **GradeScope**. Note that the PDF file of written problems needs to be concatenated with the PDF file generated from the Notebook before being submitted to GradeScope. You may access the GradeScope page through the navigation bars of the CourseWorks page. Please make sure you link the solutions and questions correctly to receive credits on Gradescope. Please include justification for all your answers - an answer with no work shown will not receive credit.
- The `hw1_code_<YOUR_UNI>.ipynb` iPython Notebook, where `<YOUR_UNI>` is your uni. This file needs to be submitted to the **CourseWorks assignment page**.

Please read the following instructions before attempting the assignment.

Collaboration Policy: You are welcome to work together with other students on the homework. However, you must write up your solutions to the assignment individually. You are also welcome to use online resources that you find helpful (e.g. tutorials, course notes, textbooks, etc.) However, directly seeking and copying/paraphrasing answers or hints for the homework questions is prohibited. If you submit an assignment that contains copied/paraphrased text or code, either from someone else or online, you will receive a 0 for the homework. Note also that if you rely too much on outside resources, you may not learn the material (the main goal!) and would likely do poorly on exams (where such resources are not available).

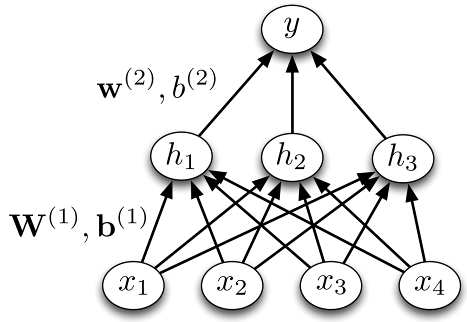
Generative AI Policy. You may ask general questions about concepts related to the homework problems. However, you may not directly ask them for hints or answers to the homework questions. For example, you should not be copying and pasting directly from the assignment handout to a chat interface. In addition, you may not directly use the output of a chatbot (or a paraphrased output) in your answers.

1 Written Problems

1.1 Hard-Coding Networks [10 pts]

The reading on multilayer perceptrons located at `https://www.cs.columbia.edu/~zemel/Class/nndl-2025/Readings/Grosse-Multilayer-Perceptrons.pdf` may be useful for this question.

We can construct hard-coded neural networks to solve algorithmic questions. In this problem, you will design a multilayer perceptron that determines if an element in a list is the k-th largest element, for a pre-specified k. The model will receive four inputs x_1, x_2, x_3, and x_4. It should output 1 if x_1 is the k-th largest element in the list, and 0 otherwise. You may assume that all elements in the input list are distinct for simplicity. Your network will have the following architecture:


y
$\mathbf{w}^{(2)}, b^{(2)}$
h_1
h_2
h_3
$\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$
x_1
x_2
x_3
x_4

In this network, $\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$ are the 3×4 weight matrix and 3-dimensional bias respectively for the hidden layer, and $\mathbf{w}^{(2)}, b^{(2)}$ are the 3-dimensional weight vector and scalar bias for the output layer. Both the hidden units and the output unit will use activation functions (you have to specify the activation functions for each layer ϕ_1, ϕ_2). Please give the weights and biases and the activation functions for this network when $k = 2$.

Note: You may find the following two activation functions useful.
1) Hard threshold activation function:

$$\phi(z) = \mathbb{I}(z > 0) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

2) Indicator activation function:

$$\phi(z) = \mathbb{I}(z \in [-1, 1]) = \begin{cases} 1 & \text{if } z \in [-1, 1] \\ 0 & \text{otherwise} \end{cases}$$

Answer

Strategy. To check whether a sequence (x_1, x_2, x_3, x_4) is superincreasing, we must satisfy the following conditions:

$$x_2 > x_1 \tag{1}$$
$$x_3 > x_1 + x_2 \tag{2}$$
$$x_4 > x_1 + x_2 + x_3 \tag{3}$$

Stated another way, our network must solve the following 3 equations:

$$x_2 - x_1 > 0 \tag{4}$$
$$x_3 - x_1 - x_2 > 0 \tag{5}$$
$$x_4 - x_3 - x_1 - x_2 > 0 \tag{6}$$

For the first layer, this can be achieved with the following weights and biases for any c > 0:

First layer:

$$\mathbf{W}^{(1)} = \begin{pmatrix} -c & c & 0 & 0 \\ -c & -c & c & 0 \\ -c & -c & -c & c \end{pmatrix}, \mathbf{b}^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

The **hard threshold** activation function should be used for this layer. Alternately, a student could use $c < 0$ and reverse these equations, with the threshold being $\phi(z) = \mathbb{I}(z < 0)$ or $\phi(z) = 1 - \mathbb{I}(z \geq 0)$

For the second layer, a solution of the following form would work for some $\varepsilon \geq 0$:

Second layer:

$$\mathbf{w}^{(2)} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}, b^{(2)} = \big(-(c_1 + c_2 + c_3 + \varepsilon)\big).$$

The correct activation function is the **indicator** function. The idea here is to do a 3-way `AND` over the inputs, to make sure that each set of first-layer conditions is satisfied. Since the indicator function checks for a number in $[-1, 1]$, you must ensure that b is sufficiently negative such that, e.g., $c_1 + c_2 + b < -1$. Some examples are $c_2 = c_2 = c_3 = 1$ and $b = -3.5$; or $c_2 = c_2 = c_3 = 2$ and $b = -6$. Note that the indicator activation function includes the values -1 and 1 in its range, so a pre-activation output value of -1 or 1 would lead to output 1—this is why answers like $c1 = c2 = c3 = 1, b = -3$ would not work!

1.2 Backpropagation

The reading on backpropagation located at `https://www.cs.columbia.edu/~zemel/Class/nndl-2025/Readings/Grosse-Backpropagation.pdf` may be useful for this question.

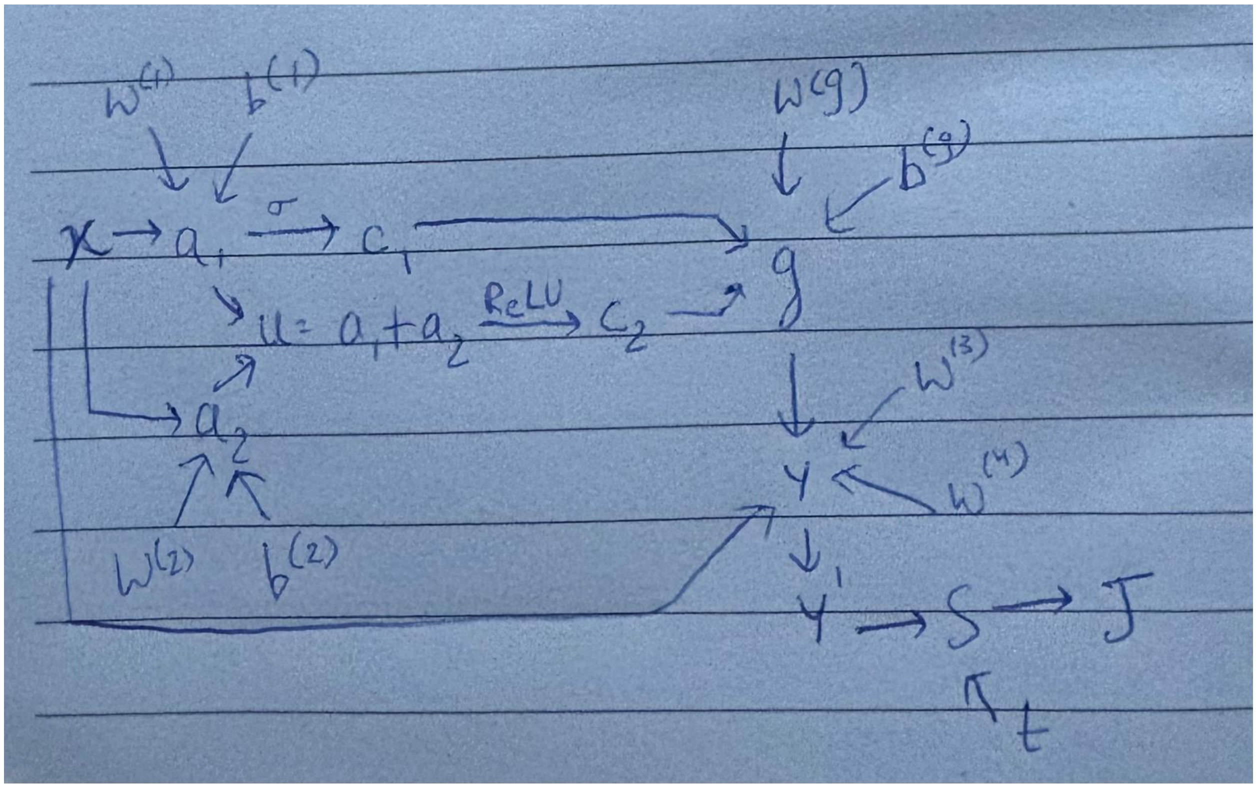
Consider a neural network defined with the following procedure:

$$
\begin{aligned}
\mathbf{a}_1 &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\
\mathbf{c}_1 &= \mathrm{ReLU}(\mathbf{a}_1) \\
\mathbf{a}_2 &= \mathbf{W}^{(2)}\mathbf{x} + \mathbf{b}^{(2)} \\
\mathbf{c}_2 &= \sigma(\mathbf{a}_2) \\
\mathbf{g} &= \mathbf{c}_1 \odot \mathbf{c}_2 \\
\mathbf{y} &= \mathbf{W}^{(3)}\mathbf{g} + \mathbf{W}^{(4)}\mathbf{x}, \\
\mathbf{y}' &= \mathrm{softmax}(\mathbf{y}) \\
\mathcal{S} &= \sum_{k=1}^{N} \mathbb{I}(t_k = 1)\log(\mathbf{y}'_k) \\
\mathcal{J} &= -\mathcal{S}
\end{aligned}
$$

for input $\mathbf{x}$ with class label t where $\mathrm{ReLU}(\mathbf{a}) = \max(\mathbf{a}, 0)$ denotes the ReLU activation function, $\sigma(\mathbf{a}) = \frac{1}{1+e^{-\mathbf{a}}}$ denotes the Sigmoid activation function, both applied elementwise, and $\mathrm{softmax}(\mathbf{y}) = \frac{\exp(\mathbf{y})}{\sum_{i=1}^{N}\exp(\mathbf{y}_i)}$. Here, $\odot$ denotes element-wise multiplication.

Computation Graph [5 pts]: Draw the computation graph relating $\mathbf{x}$, t, $\mathbf{a}_1$, $\mathbf{c}_1$, $\mathbf{a}_2$, $\mathbf{c}_2$, $\mathbf{g}$, $\mathbf{y}$, $\mathbf{y}'$, $\mathcal{S}$ and $\mathcal{J}$.

Answer



It is not necessary that the weights and biases be provided in the computation graph. Only the variables mentioned in the question mut be provided in the graph.

Backward Pass [5 pts]: 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.

Answer

$$
\begin{aligned}
\overline{\mathcal{J}} &= 1 \\
\overline{\mathcal{S}} &= \overline{\mathcal{J}} \frac{\partial \mathcal{J}}{\partial \mathcal{S}} \\
&= -\overline{\mathcal{J}} \\
\overline{\mathbf{y}'} &= \overline{\mathcal{S}} \frac{\partial \mathcal{S}}{\partial \mathbf{y}'} \\
&= \overline{\mathcal{S}}[0, \ldots, 0, 1, 0, \ldots, 0]^T \odot [\ldots, 0, \frac{1}{\mathbf{y}'_k}, 0, \ldots, 0]^T \\
&= \overline{\mathcal{S}}[\ldots, 0, \frac{1}{\mathbf{y}'_k}, 0, \ldots, 0]^T \\
&\text{where the } \frac{1}{\mathbf{y}'_k} \text{ is at the } k^{th} \text{ index.} \\
\overline{\mathbf{y}} &= \overline{\mathbf{y}'} \frac{\partial \mathbf{y}'}{\partial \mathbf{y}} \\
&= \overline{\mathbf{y}'} \text{softmax}'(\mathbf{y}) \\
\overline{\mathbf{g}} &= \overline{\mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{g}} \\
&= [\mathbf{W}^{(3)}]^\top \overline{\mathbf{y}} \\
\overline{\mathbf{c}_1} &= \overline{\mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{c}_1} \\
&= \overline{\mathbf{g}} \odot \mathbf{c}_2 \\
\overline{\mathbf{c}_2} &= \overline{\mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{c}_2} \\
&= \overline{\mathbf{g}} \odot \mathbf{c}_1 \\
\overline{\mathbf{a}_1} &= \overline{\mathbf{c}_1} \frac{\partial \mathbf{c}_1}{\partial \mathbf{a}_1} \\
&= \overline{\mathbf{c}_1} \odot \text{ReLU}'(\mathbf{a}_1) \\
&= \overline{\mathbf{c}_1} \odot \begin{cases} 0 & \mathbf{a}_{1i} < 0 \\ 1 & \mathbf{a}_{1i} > 0 \end{cases} \text{for each } i \in \{1, \ldots, K\} \\
\overline{\mathbf{a}_2} &= \overline{\mathbf{c}_2} \frac{\partial \mathbf{c}_2}{\partial \mathbf{a}_2} \\
&= \overline{\mathbf{c}_2} \odot \sigma'(\mathbf{a}_2) \\
&= \overline{\mathbf{c}_2} \odot \sigma(\mathbf{a}_2) \odot (1 - \sigma(\mathbf{a}_2)) \\
\overline{\mathbf{x}} &= \overline{\mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial \mathbf{x}} + \overline{\mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial \mathbf{x}} + \overline{\mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \\
&= [\mathbf{W}^{(1)}]^\top \overline{\mathbf{a}_1} + [\mathbf{W}^{(2)}]^\top \overline{\mathbf{a}_2} + [\mathbf{W}^{(4)}]^\top \overline{\mathbf{y}}
\end{aligned}
$$

1.3 Automatic Differentiation

Consider the function $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ where $\mathcal{L}(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{v}\mathbf{v}^\top \mathbf{x}$, and $\mathbf{v} \in \mathbb{R}^{n\times 1}$ and $\mathbf{x} \in \mathbb{R}^{n\times 1}$. Here, we will explore the relative costs of evaluating Jacobians and vector-Jacobian products. Specifically, we will study vector-Hessian products, which is a special case of vector-Jacobian products, where the Jacobian is of the gradient of our function. We denote the gradient of $\mathcal{L}$ with respect to $\mathbf{x}$ as $\mathbf{g} \in \mathbb{R}^{1\times n}$ and the Hessian of $\mathcal{L}$ w.r.t. $\mathbf{x}$ with $\mathbf{H} \in \mathbb{R}^{n\times n}$. The Hessian of $\mathcal{L}$ w.r.t. $\mathbf{x}$ is defined as:

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 \mathcal{L}}{\partial x_1^2} & \frac{\partial^2 \mathcal{L}}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 \mathcal{L}}{\partial x_1 \partial x_n} \\ \frac{\partial^2 \mathcal{L}}{\partial x_2 \partial x_1} & \frac{\partial^2 \mathcal{L}}{\partial x_2^2} & \cdots & \frac{\partial^2 \mathcal{L}}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \mathcal{L}}{\partial x_n \partial x_1} & \frac{\partial^2 \mathcal{L}}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 \mathcal{L}}{\partial x_n^2} \end{pmatrix}$$

Compute Hessian [5 pts]: Compute $\mathbf{H}$ for $n = 3$ and $\mathbf{v}^\top = [4, 2, 5]$ at $\mathbf{x}^\top = [1, 4, 3]$. In other words, write down the numbers in:

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 \mathcal{L}}{\partial x_1^2} & \frac{\partial^2 \mathcal{L}}{\partial x_1 \partial x_2} & \frac{\partial^2 \mathcal{L}}{\partial x_1 \partial x_3} \\ \frac{\partial^2 \mathcal{L}}{\partial x_2 \partial x_1} & \frac{\partial^2 \mathcal{L}}{\partial x_2^2} & \frac{\partial^2 \mathcal{L}}{\partial x_2 \partial x_3} \\ \frac{\partial^2 \mathcal{L}}{\partial x_3 \partial x_1} & \frac{\partial^2 \mathcal{L}}{\partial x_3 \partial x_2} & \frac{\partial^2 \mathcal{L}}{\partial x_3^2} \end{pmatrix}$$

Answer:

The Hessian for this loss function is computed as $\mathbf{v}\mathbf{v}^T$.

$$\mathbf{H} = \begin{pmatrix} 9 & 3 & 12 \\ 3 & 1 & 4 \\ 12 & 4 & 16 \end{pmatrix}$$

Computation Cost [5 pts]: What is the number of scalar multiplications and memory cost to compute the Hessian $\mathbf{H}$ in terms of n? You may use big $\mathcal{O}$ notation.

Answer

The number of scalar multiplications is $\mathcal{O}(n^2)$. The memory cost for storing the Hessian is also $\mathcal{O}(n^2)$.

1.3.1 Vector-Hessian Products [10 pts]

Compute $\mathbf{z} = \mathbf{H}\mathbf{y} = \mathbf{v}\mathbf{v}^\top \mathbf{y}$ where $n = 3$, $\mathbf{v}^\top = [4, 1, 2]$, $\mathbf{y}^\top = [2, 2, 1]$ 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 $\mathbf{y}$.

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 number of scalar multiplications and memory cost in 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" versus "wide".)

Answer

Part 1. For reverse-mode, we first compute $\mathbf{M} = \mathbf{v}^T\mathbf{y} = 12$. We then compute $vM = [4, 1, 2]^T[12] = [48, 12, 24]^T$.

For forward mode, we compute $\mathbf{H} = \mathbf{v}\mathbf{v}^T = \begin{pmatrix} 16 & 4 & 8 \\ 4 & 1 & 2 \\ 8 & 2 & 4 \end{pmatrix}$. Then we compute $\mathbf{H}\mathbf{y} = \begin{pmatrix} 16 & 4 & 8 \\ 4 & 1 & 2 \\ 8 & 2 & 4 \end{pmatrix} [2, 2, 1]^T = [48, 12, 24]^T$. We get the same result both ways, so $\mathbf{z}^T$ = [48, 12, 24].

Part 2. For reverse-mode, the time and memory cost are both $\mathcal{O}(mn)$. The precise number of scalar multiplications is $mn + mn + mn = 3mn$. The precise memory cost is $mn + m + mn = 2mn + m$.

For forward-mode, the time and memory cost is $\mathcal{O}(n^2 + mn)$. The precise number of scalar multiplications is $n^2 + n^2 + mn = 2n^2 + mn$. The precise memory cost is $n^2 + n + mn$.

Forward-mode is more efficient in both time and memory when $m > n$ (Z is wider). Reverse mode is more efficient in both time and memory when $m < n$ (Z is taller).

While costs are often reported in big-O notation, reasoning about when forward mode is strictly better than reverse-mode may require referring to the exact cost of these operations.