Due Date : March 1st (11pm), 2021

Instructions

- *For all questions, show your work!*
- *Use LaTeX and the template we provide when writing your answers. You may reuse most of the notation shorthands, equations and/or tables. See the assignment policy on the course website for more details.*
- *Submit your answers electronically via Gradescope.*
- ***TAs for this assignment are Sébastien Lachapelle and Arian Hosseini.***

Question 1 (4-4-4)**.** Using the following definition of the derivative and the definition of the Heaviside step function :

$$\frac{d}{dx}f(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon} \qquad H(x) = \begin{cases} 1 & \text{if } x > 0 \\ \frac{1}{2} & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases}$$

1.1 Show that the derivative of the rectified linear unit $g(x) = \max\{0, x\}$, **wherever it exists**, is equal to the Heaviside step function.

The derivative of a function at point x_0 exists if both of the right and left limits of $\frac{d}{dx}f(x)$ exist and are **equal**, i.e. :

1. $\lim_{x \to x_0^-} \frac{df(x)}{dx}$ exists
2. $\lim_{x \to x_0^+} \frac{df(x)}{dx}$ exists
3. $\lim_{x \to x_0^-} \frac{df(x)}{dx} = \lim_{x \to x_0^+} \frac{df(x)}{dx}$

Thus we consider the existence of this derivative in 3 regions :

- $x_0 < 0$, first the left limit of the derivative :

$$\lim_{\epsilon \to 0^-} \frac{g(x_0+\epsilon) - g(x_0)}{\epsilon} \overset{g(x)=\max\{0,x\}}{=} \lim_{\epsilon \to 0^-} \frac{\max\{0, x_0+\epsilon\} - \max\{0, x_0\}}{\epsilon}$$

But since $x_0 < 0$, we're in the dead region of ReLU and thus $\max\{0, x_0+\epsilon\} = \max\{0, x_0\} = 0$, and therefore :

$$\lim_{\epsilon \to 0^-} \frac{\max\{0, x_0+\epsilon\} - \max\{0, x_0\}}{\epsilon} = \lim_{\epsilon \to 0^-} \frac{0-0}{\epsilon} = 0$$

So the left limit of the derivative exists. Now the right limit :

$$\lim_{\epsilon \to 0^+} \frac{g(x_0+\epsilon) - g(x_0)}{\epsilon} \overset{g(x)=\max\{0,x\}}{=} \lim_{\epsilon \to 0^+} \frac{\max\{0, x_0+\epsilon\} - \max\{0, x_0\}}{\epsilon}$$

But again since $x_0 < 0$, we're in the dead region of ReLU and thus $\max\{0, x_0 + \epsilon\} = \max\{0, x_0\} = 0$, and therefore :

$$\lim_{\epsilon \to 0^+} \frac{\max\{0, x_0 + \epsilon\} - \max\{0, x_0\}}{\epsilon} = \lim_{\epsilon \to 0^+} \frac{0 - 0}{\epsilon} = 0$$

Hence, both the right and left limits of the derivative exist and are equal, so the derivative for any $x < 0$ **exists** and is **equal to 0**.

- $x_0 = 0$, first the left limit of the derivative :

$$\lim_{\epsilon \to 0^-} \frac{g(0 + \epsilon) - g(0)}{\epsilon} \overset{g(x)=\max\{0,x\}}{=} \lim_{\epsilon \to 0^-} \frac{\max\{0, 0 + \epsilon\} - \max\{0, 0\}}{\epsilon}$$

But since $x_0 = 0$, $0 + \epsilon^-$ would be in the dead region of ReLU and thus $\max\{0, 0 + \epsilon^-\} = \max\{0, 0\} = 0$, and therefore :

$$\lim_{\epsilon \to 0^-} \frac{\max\{0, 0 + \epsilon\} - \max\{0, 0\}}{\epsilon} = \lim_{\epsilon \to 0^-} \frac{0 - 0}{\epsilon} = 0$$

So the left limit of the derivative exists. Now the right limit :

$$\lim_{\epsilon \to 0^+} \frac{g(0 + \epsilon) - g(0)}{\epsilon} \overset{g(x)=\max\{0,x\}}{=} \lim_{\epsilon \to 0^+} \frac{\max\{0, 0 + \epsilon\} - \max\{0, 0\}}{\epsilon}$$

But now since $x_0 = 0$, $0 + \epsilon^+$ falls in the active region of ReLU and thus $\max\{0, 0 + \epsilon\} = \epsilon, \max\{0, 0\} = 0$, and therefore :

$$\lim_{\epsilon \to 0^+} \frac{\max\{0, 0 + \epsilon\} - \max\{0, 0\}}{\epsilon} = \lim_{\epsilon \to 0^+} \frac{\epsilon - 0}{\epsilon} = 1$$

Now both the right and left limits of the derivative exist **but are NOT equal**, so the derivative at $x = 0$ **does NOT exist**.

- $x_0 > 0$, first the left limit of the derivative :

$$\lim_{\epsilon \to 0^-} \frac{g(x_0 + \epsilon) - g(x_0)}{\epsilon} \overset{g(x)=\max\{0,x\}}{=} \lim_{\epsilon \to 0^-} \frac{\max\{0, x_0 + \epsilon\} - \max\{0, x_0\}}{\epsilon}$$

But since $x_0 > 0$, $x_0 + \epsilon^-, x_0$ fall in the active region of ReLU and thus $\max\{0, x_0 + \epsilon\} = x_0 + \epsilon, \max\{0, x_0\} = x_0$, and therefore :

$$\lim_{\epsilon \to 0^-} \frac{\max\{0, x_0 + \epsilon\} - \max\{0, x_0\}}{\epsilon} = \lim_{\epsilon \to 0^-} \frac{x_0 + \epsilon - x_0}{\epsilon} = 1$$

So the left limit of the derivative exists. Now the right limit :

$$\lim_{\epsilon \to 0^+} \frac{g(x_0 + \epsilon) - g(x_0)}{\epsilon} \overset{g(x)=\max\{0,x\}}{=} \lim_{\epsilon \to 0^+} \frac{\max\{0, x_0 + \epsilon\} - \max\{0, x_0\}}{\epsilon}$$

Again since $x_0 > 0$, $x_0 + \epsilon^+, x_0$ fall in the active region of ReLU and thus $\max\{0, x_0 + \epsilon\} = x_0 + \epsilon, \max\{0, x_0\} = x_0$, and therefore :

$$\lim_{\epsilon \to 0^+} \frac{\max\{0, x_0 + \epsilon\} - \max\{0, x_0\}}{\epsilon} = \lim_{\epsilon \to 0^+} \frac{x_0 + \epsilon - x_0}{\epsilon} = 1$$

Now both the right and left limits of the derivative exist and are equal, so the derivative at any $x > 0$ **exists** and is **equal to 1**.

So to summarize :

$$H(x) = \begin{cases} 1 & \text{if } x > 0 \\ \frac{1}{2} & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases} \qquad \frac{dg(x)}{dx} = \begin{cases} 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases}$$

It's clear that wherever the derivative of $g(x)$ exists, it's equal to the Heaviside step function.

1.2 Give two alternative definitions of $g(x)$ using $H(x)$.

From the previous part we've seen that the step function relates to $g(x)$ by a derivative operation, so the natural thing that comes to mind, is integration. So the first alternative would be :

$$g(x) = \int_{-\infty}^{x} H(x)dx$$

It's clear that for any $x < 0$ this integral amounts to zero. For $x = 0$ again it's zero since a single point's contribution to the integral would be zero. For any $x > 0$ the result of the integral would be

$$\int_{-\infty}^{x} H(x)dx = \int_{0}^{x} H(x)dx = \int_{0}^{x} 1dx = x$$

So we have that :

$$\int_{-\infty}^{x} H(x)dx = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases} \qquad g(x) = \max\{0, x\} = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases}$$

So this alternative is valid.

For the second one, since step function's dead side multiplied by anything results zero, we can think of multiplication and thus come up with $xH(x)$. For all $x < 0$ it'll be zero, for $x = 0$ it'll be zero because of the x itself, and for $x > 0$ it'll be $1 \times x = x$, so :

$$x \times H(H) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases} \qquad g(x) = \max\{0, x\} = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases}$$

So this second alternative is also valid.

1.3 Show that $H(x)$ can be well approximated by the sigmoid function $\sigma(x) = \frac{1}{1+e^{-kx}}$ asymptotically (i.e for large k), where k is a parameter.

To show that we define the difference $D(x) = H(x) - \sigma(x)$, evaluate the difference in different regions of x and take the limit when $k \to \infty$:

$$D(x) = H(x) - \sigma(x) = \begin{cases} 1 - \frac{1}{1+e^{-kx}} = \frac{e^{-kx}}{1+e^{-kx}} & \text{if } x > 0 \\ 0 - \frac{1}{1+e^{-kx}} = -\frac{1}{1+e^{-kx}} & \text{if } x < 0 \\ \frac{1}{2} - \frac{1}{2} = 0 & \text{if } x = 0 \end{cases}$$

$$\lim_{k\to\infty} D(x) = \begin{cases} \lim_{k\to\infty} \frac{e^{-kx}}{1+e^{-kx}} & \text{if } x > 0 \\ \lim_{k\to\infty} -\frac{1}{1+e^{-kx}} & \text{if } x < 0 \\ \lim_{k\to\infty} 0 & \text{if } x = 0 \end{cases} = \begin{cases} \frac{0}{1+0} & \text{if } x > 0 \\ -\frac{1}{1+\infty} & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases}$$

Thus,

$$\lim_{k\to\infty} D(x) = H(x) - \sigma(x) = \begin{cases} 0 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases}$$

So the difference asymptotically approaches zero for all $x \in \mathbb{R}$, therefore we can say that $H(x)$ is asymptotically approximated by $\sigma(x)$ (asymptotic in k).

Question 2 (3-4-4-4)**.** Recall the definition of the softmax function : $S(\boldsymbol{x})_i = e^{\boldsymbol{x}_i} / \sum_j e^{\boldsymbol{x}_j}$.

2.1 Show that softmax is translation-invariant, that is : $S(\boldsymbol{x} + c) = S(\boldsymbol{x})$, where c is a scalar constant.

$$S(\boldsymbol{x} + c)_i = \frac{e^{x_i + c}}{\sum_j e^{x_j + c}} = \frac{\cancel{e^c} e_i^x}{\cancel{e^c} \sum_j e_j^x} = S(\boldsymbol{x})_i$$

Since this is true for all i, then we have that $S(\boldsymbol{x}+c) = S(\boldsymbol{x})$, so softmax is translation-invariant.

2.2 Let $\boldsymbol{x}$ be a 2-dimensional vector. One can represent a 2-class categorical probability using softmax $S(\boldsymbol{x})$. Show that $S(\boldsymbol{x})$ can be reparameterized using sigmoid function, i.e. $S(\boldsymbol{x}) = [\sigma(z), 1 - \sigma(z)]^\top$ where z is a scalar function of $\boldsymbol{x}$.

$$S(\boldsymbol{x})_1 = \frac{e_1^x}{e_1^x + e_2^x}, S(\boldsymbol{x})_2 = \frac{e_2^x}{e_1^x + e_2^x}$$

Or equivalently (dividing by e_1^x, e_2^x respectively) :

$$S(\boldsymbol{x})_1 = \frac{1}{1 + e^{-(x_1 - x_2)}}, S(\boldsymbol{x})_2 = \frac{1}{1 + e^{x_1 - x_2}}$$

Now if we define $z := x_1 - x_2$ then we have :

$$S(\boldsymbol{x}) = [\frac{1}{1 + e^{-z}}, \frac{1}{1 + e^{z}}]^\top = [\sigma(z), \sigma(-z)]^\top$$

But note that $\sigma(-z) = \frac{1}{1+e^z} = \frac{e^{-z}}{1+e^{-z}} = \frac{-1+1+e^{-z}}{1+e^{-z}} = 1 - \sigma(z)$, thus we can replace $\sigma(-z)$ by $1 - \sigma(z)$:

$$S(\boldsymbol{x}) = [\sigma(z), 1 - \sigma(z)]^\top$$

2.3 Let $\boldsymbol{x}$ be a K-dimensional vector ($K \geq 2$). Show that $S(\boldsymbol{x})$ can be represented using $K-1$ parameters, i.e. $S(\boldsymbol{x}) = S([0, y_1, y_2, ..., y_{K-1}]^\top)$, where y_i is a scalar function of $\boldsymbol{x}$ for $i \in \{1, ..., K-1\}$.

In 2.1 we showed that softmax is translation-invariant, so if we let $c = -(\boldsymbol{x})_0$, then we know that due to translation-invariance $S(\boldsymbol{x}+c) = S(\boldsymbol{x})$, but $S(\boldsymbol{x}+c) = S([x_0-x_0, x_1-x_0, \ldots, x_{K-1}-x_0]^\top)$ which is equal to $S([0, y_1, \ldots, y_{K-1}]^\top)$, where $y_i = x_i - x_0$ are scalar functions of $\boldsymbol{x}$ for $i \in \{1, \ldots, K-1\}$. Thus we've showed that $S(\boldsymbol{x})$ can be represented using $K-1$ parameters.

2.4 Show that the Jacobian of the softmax function $J_{\text{softmax}}(\boldsymbol{x})$ can be expressed as : $\mathbf{Diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^\top$, where $\boldsymbol{p} = S(\boldsymbol{x})$.

If we have a vector valued function $f(\boldsymbol{x}) = [f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})]^\top$ like the softmax function, then the Jacobian definition is ($\boldsymbol{x} = [x_1, \ldots, x_m]^\top$) :

$$J_f(x) = \begin{pmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_K(x)}{\partial x_1} & \cdots & \frac{\partial f_K(x)}{\partial x_m} \end{pmatrix}$$

And

$$f(\boldsymbol{x}) = [\frac{e^{x_1}}{\sum_j e^{x_j}}, \frac{e^{x_2}}{\sum_j e^{x_j}}, \ldots, \frac{e^{x_m}}{\sum_j e^{x_j}}]^\top$$

Therefore if we plug in the softmax function f into the Jacobian definition we get :

$$J_f(x) = \begin{pmatrix} \frac{e^{x_1}\sum_j e^{x_j} - e^{x_1}e^{x_1}}{(\sum_j e^{x_j})^2} & \frac{-e^{x_1}e^{x_2}}{(\sum_j e^{x_j})^2} & \cdots & \cdots & \frac{-e^{x_1}e^{x_m}}{(\sum_j e^{x_j})^2} \\ \frac{-e^{x_1}e^{x_2}}{(\sum_j e^{x_j})^2} & \frac{e^{x_1}\sum_j e^{x_j} - e^{x_2}e^{x_2}}{(\sum_j e^{x_j})^2} & \frac{-e^{x_2}e^{x_3}}{(\sum_j e^{x_j})^2} & \cdots & \frac{-e^{x_2}e^{x_m}}{(\sum_j e^{x_j})^2} \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ \frac{-e^{x_1}e^{x_m}}{(\sum_j e^{x_j})^2} & \cdots & \cdots & \cdots & \frac{e^{x_m}\sum_j e^{x_j} - e^{x_m}e^{x_m}}{(\sum_j e^{x_j})^2} \end{pmatrix}$$

As representatives, consider J_{11}, J_{12}, we'll simplify them and the rest follows the similar procedure :

$$J_{11} = \frac{e^{x_1}\sum_j e^{x_j} - e^{x_1}e^{x_1}}{(\sum_j e^{x_j})^2} = \frac{e^{x_1}\cancel{\sum_j e^{x_j}}}{(\sum_j e^{x_j})\cancel{^2}} - (\frac{e^{x_1}}{\sum_j e^{x_j}})^2 = S(\boldsymbol{x})_1 - S^2(\boldsymbol{x})_1$$

$$J_{12} = \frac{-e^{x_1}e^{x_2}}{(\sum_j e^{x_j})^2} = -\frac{e^{x_1}}{\sum_j e^{x_j}}\frac{e^{x_2}}{\sum_j e^{x_j}} = -S(\boldsymbol{x})_1 S(\boldsymbol{x})_2$$

Now we can simplify the Jacobian :

$$J_f(x) = \begin{pmatrix} S(\boldsymbol{x})_1 - S^2(\boldsymbol{x})_1 & -S(\boldsymbol{x})_1 S(\boldsymbol{x})_2 & \cdots & \cdots & -S(\boldsymbol{x})_1 S(\boldsymbol{x})_m \\ -S(\boldsymbol{x})_1 S(\boldsymbol{x})_2 & S(\boldsymbol{x})_2 - S^2(\boldsymbol{x})_2 & \cdots & \cdots & -S(\boldsymbol{x})_2 S(\boldsymbol{x})_m \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ -S(\boldsymbol{x})_1 S(\boldsymbol{x})_m & \cdots & \cdots & \cdots & S(\boldsymbol{x})_m - S^2(\boldsymbol{x})_m \end{pmatrix}$$

Decomposing it into the subtraction of two matrices :

$$= \begin{pmatrix} S(\boldsymbol{x})_1 & & & O \\ & S(\boldsymbol{x})_2 & & \\ & & \ddots & \\ O & & & S(\boldsymbol{x})_m \end{pmatrix} - \begin{pmatrix} S(\boldsymbol{x})_1 \\ S(\boldsymbol{x})_2 \\ \vdots \\ S(\boldsymbol{x})_m \end{pmatrix} \begin{pmatrix} S(\boldsymbol{x})_1 & S(\boldsymbol{x})_2 & \dots & S(\boldsymbol{x})_m = \end{pmatrix}$$
$$= \text{Diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^\top$$

where $\boldsymbol{p} = S(\boldsymbol{x})$.

Question 3 (2-6-6-4)**.** Consider the differentiable functions $f : \mathbb{R}^\ell \to \mathbb{R}^m$, $g : \mathbb{R}^m \to \mathbb{R}^n$, and $h : \mathbb{R}^n \to \mathbb{R}^o$. Let $F : \mathbb{R}^\ell \to \mathbb{R}^o$ be the composition of these functions, i.e. $F = h \circ g \circ f$. For this question, denote the Jacobian matrix of F evaluated at $x \in \mathbb{R}^\ell$ as $J_F(x) \in \mathbb{R}^{o \times \ell}$, and analogously for any other function.

3.1 Using the chain rule, express $J_F(x)$ using J_f, J_g and J_h. Your answer should be in matrix form. In your expression, make sure it is clear at which point each Jacobian matrix is evaluated.

Let's first take a look at the definition of $J_F(x)$, the Jacobian evaluated at point $\boldsymbol{x}$ where $F(x)$ is a vector valued function $F(x) = [F_1(x), F_2(x), \dots, F_o(x)]^\top$:

$$J_F(x) = \begin{pmatrix} \partial F_1/\partial x_1 & \dots & \partial F_1/\partial x_l \\ \vdots & \ddots & \vdots \\ \partial F_o/\partial x_1 & \dots & \partial F_o/\partial x_l \end{pmatrix}_{o \times l}$$

Also note that f, g, h are all vector-valued functions :

$$\mathbf{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{pmatrix}_{m \times 1} \qquad \mathbf{g} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix}_{n \times 1} \qquad \mathbf{h} = \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_o \end{pmatrix}_{o \times 1}$$

For each of these partials $\partial F_u/\partial x_v$ we can use the chain rule. Also note that $\mathbf{F}(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x})))$, (any letter in bold denotes either a vector, or a vector-valued function) so for each partial we have :

$$\frac{\partial F_u}{\partial x_v}|_{\boldsymbol{x}} = \partial h_u(\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x})))/\partial x_v = \sum_{j=1}^{n} (\nabla_g h_u)_j|_{\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x}))} \frac{\partial g_j}{\partial x_v}|_{\boldsymbol{x}}$$

Or in matrix form :

$$J_F(x)|_{\boldsymbol{x}} = \underbrace{\begin{pmatrix} (\nabla_g h_1)_1 & (\nabla_g h_1)_2 & \dots & (\nabla_g h_1)_n \\ (\nabla_g h_2)_1 & (\nabla_g h_2)_2 & \dots & (\nabla_g h_2)_n \\ \vdots & \vdots & \ddots & \vdots \\ (\nabla_g h_o)_1 & \dots & \dots & (\nabla_g h_o)_n \end{pmatrix}_{o \times n}|_{\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x}))}}_{J_h(\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x})))} \times \begin{pmatrix} \partial g_1/\partial x_1 & \partial g_1/\partial x_2 & \dots & \partial g_1/\partial x_l \\ \partial g_2/\partial x_1 & \partial g_2/\partial x_2 & \dots & \partial g_2/\partial x_l \\ \vdots & \vdots & \ddots & \vdots \\ \partial g_n/\partial x_1 & \dots & \dots & \partial g_n/\partial x_l \end{pmatrix}_{n \times l}|_{\boldsymbol{x}}$$

Now we should simplify the partials $\partial g_u/\partial x_v$ to get them in terms of J_g, J_f. Following a similar procedure we observe that :

$$\frac{\partial(g_u(\boldsymbol{f}(\boldsymbol{x})))}{\partial x_v}|_{\boldsymbol{x}} = \sum_{i=1}^{m} (\nabla_f g_u)_i|_{\boldsymbol{f}(\boldsymbol{x})} \frac{\partial f_i}{\partial x_v}|_{\boldsymbol{x}}$$

Or in matrix form :

$$J_g(x)|_{\boldsymbol{x}} = \underbrace{\begin{pmatrix} (\nabla_f g_1)_1 & (\nabla_f g_1)_2 & \dots & (\nabla_f g_1)_m \\ (\nabla_f g_2)_1 & (\nabla_f g_2)_2 & \dots & (\nabla_f g_2)_m \\ \vdots & \vdots & \ddots & \vdots \\ (\nabla_f g_n)_1 & \dots & \dots & (\nabla_f g_n)_m \end{pmatrix}_{n\times m}^{|_{\boldsymbol{f}(\boldsymbol{x})}}}_{J_g(\boldsymbol{f}(\boldsymbol{x}))} \times \underbrace{\begin{pmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 & \dots & \partial f_1/\partial x_l \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 & \dots & \partial f_2/\partial x_l \\ \vdots & \vdots & \ddots & \vdots \\ \partial f_m/\partial x_1 & \dots & \dots & \partial f_m/\partial x_l \end{pmatrix}_{m\times l}^{|_{\boldsymbol{x}}}}_{J_f(\boldsymbol{x})}$$

Plugging these back to $J_F(x)|_{\boldsymbol{x}}$ we get the following matrix multiplication :

$$J_F(x)|_{\boldsymbol{x}} = J_h(\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x})))J_g(\boldsymbol{f}(\boldsymbol{x}))J_f(\boldsymbol{x})$$

It means that $J_h(\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x})))$ is the Jacobian of h evaluated at $\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x}))$, $J_g(\boldsymbol{f}(\boldsymbol{x}))$ is the Jacobian of g evaluated at $\boldsymbol{f}(\boldsymbol{x})$, $J_f(\boldsymbol{x})$ is the Jacobian of f evaluated at $\boldsymbol{x}$.

3.2 Provide a simple pseudo code which computes $J_F(x)$ using *forward mode accumulation* (Section 6.5.9 in DL book) given x. You can call the functions f, g, h, J_f, J_g and J_h **only once each** (to maximize efficiency). You can call the function `matmul`$(\cdot,\cdot)$ which performs matrix multiplication (no limit on the number of calls). Your pseudo code should also return $F(x)$.

In the forward mode accumulation, we traverse inside out and at each point both the function and the derivative are computed, hence it doesn't require storing any value in the intermediate steps as opposed to the reverse mode where we store the computations of the forward pass for computing the Jacobians. So we start from computing $f(x), J_f(x)$ and move outwards, similar to multiplying from right to left (numbers below show the ordering in matrix multiplication) :

$$J_F(x)|_{\boldsymbol{x}} = \underbrace{J_h(\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x})))\underbrace{J_g(\boldsymbol{f}(\boldsymbol{x}))J_f(\boldsymbol{x})}_{\#1}}_{\#2}$$

Thus the pseudo code would be the following :

Algorithm 1: Forward Mode Accumulation

Result: Computes the Jacobian and the value of the composite function using forward mode accumulation

Select the point $\boldsymbol{x}$ at which you want to compute the composite function Jacobian;
Define two temporary variables for storing evaluations and Jacobians called `f` and `J`;
call $f(\boldsymbol{x})$, compute $J_f(\boldsymbol{x})$, store them temporarily in `f`, `J`, respectively;
Using the stored values, call $g(f(\boldsymbol{x}))$ and `matmul`$(J_g(f), J_f)$ as below :
`J` $\leftarrow$ `matmul`$(J_g($`f`$)$, `J`$)$;
`f` $\leftarrow g($`f`$)$;
Using the stored values, call $h(g(f(\boldsymbol{x})))$ and `matmul`$(J_h(g), J_g)$ as below :
`J` $\leftarrow$ `matmul`$(J_h($`f`$)$, `J`$)$;
`f` $\leftarrow h($`f`$)$;
Return `f`, `J` as the function and the Jacobian evaluation, respectively;

3.3 Provide a simple pseudo code which computes $J_F(x)$ in *reverse mode accumulation* (Section 6.5.9 in DL book) given x. You can call the functions f, g, h, J_f, J_g and J_h **only once each** (to maximize efficiency). You can call the function $\texttt{matmul}(\cdot,\cdot)$ which performs matrix multiplication (no limit on the number of calls). Your pseudo code should also return $F(x)$.

For this mode, we should do the forward pass, but *keep* the values at intermediate steps, i.e. we do the forward pass and keep all values for $f(\boldsymbol{x}), g(f(\boldsymbol{x})), h(g(f(\boldsymbol{x})))$ and do the matrix multiplication from left to right :

$$J_F(x)|_{\boldsymbol{x}} = \underbrace{\underbrace{J_h(\boldsymbol{g}(\boldsymbol{f}(\boldsymbol{x})))J_g(\boldsymbol{f}(\boldsymbol{x}))}_{\#1} J_f(\boldsymbol{x})}_{\#2}$$

Thus the pseudo code would be the following :

Algorithm 2: Reverse Mode Accumulation

Result: Computes the Jacobian and the value of the composite function using **reverse** mode accumulation

Select the point $\boldsymbol{x}$ at which you want to compute the composite function Jacobian;

Define temporary variables for storing evaluations of $f(\boldsymbol{x}), g(f(\boldsymbol{x})), h(f(\boldsymbol{x}))$ called $\texttt{f}, \texttt{g}, \texttt{h}$, and $\texttt{J}$ for the total Jacobian;

Do the forward pass, i.e. $\texttt{f} \leftarrow f(\boldsymbol{x})$, then $\texttt{g} \leftarrow g(\texttt{f})$, then $\texttt{h} \leftarrow h(\texttt{g})$;

Using the stored values, call $J_h(g), J_g(f)$ and compute their matrix product, i.e. $\texttt{J} \leftarrow \texttt{matmul}(J_h(\texttt{g}), J_g(\texttt{f}))$;

Using the stored values, call $J_f(\boldsymbol{x})$ and compute the matrix product $\texttt{matmul}(J_{hg}, J_f(\boldsymbol{x}))$, i.e. :$\texttt{J} \leftarrow \texttt{matmul}(\texttt{J}, J_f(\boldsymbol{x}))$;

Return $\texttt{h}, \texttt{J}$ as the function and the Jacobian evaluation, respectively;

3.4 Assume evaluating f and J_f cost $O(\ell m)$, evaluating g and J_g cost $O(mn)$ and evaluating h and J_h cost $O(no)$. What is the time complexity of your forward mode pseudo code ? your reverse mode pseudo code ?

Forward Mode :

Going through the pseudo code, it takes $O(\ell m)$ to evaluate $f(\boldsymbol{x})$ and $J_f(\boldsymbol{x})$, $O(mn)$ to evaluate g and J_g (using obtained $f(\boldsymbol{x}), J_f(\boldsymbol{x})$), takes $O(nm\ell)$ to compute $\texttt{matmul}(J_g(f), J_f)$, takes $O(no)$ to evaluate h and J_h (using obtained $g(f(\boldsymbol{x})), J_g(f(\boldsymbol{x}))$), takes $O(on\ell)$ to compute $\texttt{matmul}(J_h(g), J_g)$. So overall :

$$\text{time complexity} = O(\ell m) + O(mn) + O(nm\ell) + O(on) + O(on\ell)$$

Reverse Mode :

Going through the pseudo code, for the forward pass it takes $O(\ell m)$ to evaluate $f(\boldsymbol{x})$, $O(mn)$ to evaluate g (using obtained $f(\boldsymbol{x})$), $O(no)$ to evaluate h (using obtained $g(f(\boldsymbol{x}))$). For the backward pass, it takes $O(no)+O(mn)$ to evaluate $J_h(g)$, $J_g(f)$ (using obtained $g(f(\boldsymbol{x})), f(\boldsymbol{x})$), $O(onm)$ to compute $J_{hg} = \texttt{matmul}(J_h(g), J_g)$, $O(\ell m)$ to evaluate J_f, $O(om\ell)$ to compute $J_F = \texttt{matmul}(J_{hg}, J_f)$. So overall :

$$\begin{aligned}\text{time complexity} &= O(\ell m) + O(mn) + O(no) + O(no) + O(mn) + O(onm) + O(\ell m) + O(om\ell) \\ &= O(\ell m) + O(mn) + O(no) + O(onm) + O(om\ell)\end{aligned}$$

If we compare the time complexities, we observe the difference is in terms with three elements and those are the dominating factors, for instance consider the forward mode, $O(lm), O(mn)$ appear in $O(nml)$, and $O(on)$ appears in $O(onl)$, so basically these terms with two elements contribute to a factor of $O(1)$ in the terms with three elements, and thus they're absorbed.
So approximately for the forward pass we have $O(nml) + O(onl) = O(nl(m+o))$, and for the backward we have $O(om(l+n))$. In neural networks we often have very large inputs (1000×1000 images) but number of outputs are far less than the input dimension, i.e. on the order of 1000. So $o \ll l$, in that case the time complexity of the forward mode would approximate to $O(nl(m+o)) \approx O(nlm)$, whereas for the reverse mode it would be $O(om(l+n)) \approx O(olm)$ and since $o \ll l$ it justifies why we use some specific kind of reverse mode (backprop) with neural nets. However it should be noted that the memory complexity of the reverse mode is significantly larger than the forward mode, since it requires keeping all intermediate values (as can be seen by the larger number of temporary values defined and kept in alg. 2), whereas in the forward mode we can discard the values *as we pass* and evaluate the next layer outputs and Jacobians.

Question 4 (5)**.** Compute the *full*, *valid*, and *same* convolution (with kernel flipping) for the following 1D matrices : $\big[5, 6, 7, 8\big] * \big[3, 0, 1\big]$

We first note that here kernels have to be *flipped* before being applied. Second, we know from the class that the output dimension for the result of a convolution is derived from the following :

$$o = \lfloor \frac{i + 2p - k}{s} \rfloor + 1$$

Where i is the input size, p is the padding (determined by the convolution mode), k is the kernel size, s is the stride, which is 1 for all of the cases. Now we can proceed.
For the *valid* convolution which there's no padding, the output size would be $o = \lfloor \frac{i+2p-k}{s} \rfloor + 1 = \lfloor \frac{4+2\times0-3}{1} \rfloor + 1 = 2$. Hence we have (note that kernels have been flipped) :

$$\begin{aligned} y[0] &= [5, 6, 7] \cdot [1, 0, 3] = 5 + 21 = 26 \\ y[1] &= [6, 7, 8] \cdot [1, 0, 3] = 6 + 24 = 30 \\ &\implies y = [26, 30]^\top \end{aligned}$$

For the *same* convolution which there's 1 zero padding to each side of the sequence, the output size would be $o = \lfloor \frac{i+2p-k}{s} \rfloor + 1 = \lfloor \frac{4+2\times1-3}{1} \rfloor + 1 = 4$. Hence we have (note that kernels have been flipped) :

$$\begin{aligned} y[0] &= [0, 5, 6] \cdot [1, 0, 3] = 18 \\ y[1] &= [5, 6, 7] \cdot [1, 0, 3] = 26 \\ y[2] &= [6, 7, 8] \cdot [1, 0, 3] = 30 \\ y[3] &= [7, 8, 0] \cdot [1, 0, 3] = 7 \\ &\implies y = [18, 26, 30, 7]^\top \end{aligned}$$

For the *full* convolution which there are 2 zero paddings to each side of the sequence (so that all elements of the kernel touch all elements of the input), the output size would be $o = \lfloor \frac{i+2p-k}{s} \rfloor + 1 =$

$\lfloor \frac{4+2\times 2-3}{1} \rfloor + 1 = 6$. Hence we have (note that kernels have been flipped) :

$$\begin{aligned} y[0] &= [0,0,5] \cdot [1,0,3] = 15 \\ y[1] &= [0,5,6] \cdot [1,0,3] = 18 \\ y[2] &= [5,6,7] \cdot [1,0,3] = 26 \\ y[3] &= [6,7,8] \cdot [1,0,3] = 30 \\ y[4] &= [7,8,0] \cdot [1,0,3] = 7 \\ y[5] &= [8,0,0] \cdot [1,0,3] = 8 \\ \implies y &= [15,18,26,30,7,8]^\top \end{aligned}$$

Question 5 (5-5)**.** Consider a convolutional neural network. Assume the input is a colorful image of size 128×128 in the RGB representation. The first layer convolves 32 8×8 kernels with the input, using a stride of 2 and a zero-padding of 3 (three zeros on each side). The second layer downsamples the output of the first layer with a 2×2 non-overlapping max pooling. The third layer convolves 64 3×3 kernels with a stride of 1 and a zero-padding of size 1 on each border.

5.1 What is the dimensionality of the output of the last layer, i.e. the number of scalars it contains ?

we know that the output dimension for the result of a convolution is derived from the following :

$$o = \lfloor \frac{i + 2p - k}{s} \rfloor + 1$$

Where i is the input size, p is the padding (determined by the convolution mode), k is the kernel size, s is the stride. Input layer has 3 channels each of which is 128×128. Convolving it with 8×8 kernels with zero padding of 3 and stride 2, gives $o = \lfloor \frac{i+2p-k}{s} \rfloor + 1 = \lfloor \frac{128+2\times 3-8}{2} \rfloor + 1 = 64$, so in the second layer, we have 32 channels (corresponding to 32 8×8 kernels) each of which being 64×64. Then after second layer, non-overlapping max pooling of size 2×2 would divide each dimension by 2 so in the second layer we have again 32 channels (pooling doesn't affect the number of channels) each of which now being 32×32. Then after the third layer's convolution, we have 64 channels of size $o \times o$ where $o = \lfloor \frac{i+2p-k}{s} \rfloor + 1 = \lfloor \frac{32+2\times 1-3}{1} \rfloor + 1 = 32$. So the dimensionality of the last layer would be $64 \times 32 \times 32 = 65536$

5.2 Not including the biases, how many parameters are needed for the last layer ?

We should note that each kernel in the last layer is fully connected to *all* channels of the layer before it. So each kernel of size 3×3 corresponds to :

$$\#(\text{prev. layer channels}) \times \text{kernel size} \times \text{kernel size}$$

number of parameters. Because there are 64 of these kernels in the third layer, the total number of weights for this layer would be $64 \times 32 \times 3 \times 3 = 18432$

Question 6 (1-4-5-1-4-5)**.** Let us use the notation $*$ and $\tilde{*}$ to denote the valid and full convolution operator **without kernel flipping**, respectively. The operations are defined as

$$\text{valid} : (\boldsymbol{x} * \boldsymbol{w})_n = \sum_{j=1}^{k} x_{n+j-1} w_j \tag{1}$$

$$\text{full} : (\boldsymbol{x} \,\tilde{*}\, \boldsymbol{w})_n = \sum_{j=1}^{k} x_{n+j-k} w_j \ , \tag{2}$$

where k is the size of the kernel $\boldsymbol{w}$. As a convention, the value of a vector indexed "out-of-bound" is zero, e.g. if $\boldsymbol{x} \in \mathbb{R}^d$, then $x_i = 0$ for $i < 1$ and $i > d$. We define the flip operator which reverse the ordering of the components of a vector, i.e. $\text{flip}(\boldsymbol{w})_j = w_{k-j+1}$.

Consider a convolutional network with 1-D input $\boldsymbol{x} \in \mathbb{R}^d$. Its first and second convolutional layers have kernel $\boldsymbol{w}^1 \in \mathbb{R}^{k_1}$ and $\boldsymbol{w}^2 \in \mathbb{R}^{k_2}$, respectively. Assume $k_1 < d$ and $k_2 < d$. The network is specified as follows :

$$\boldsymbol{a}^1 \leftarrow \boldsymbol{x} * \boldsymbol{w}^1 \text{ (valid convolution)} \tag{3}$$
$$\boldsymbol{h}^1 \leftarrow \text{ReLU}(\boldsymbol{a}^1) \tag{4}$$
$$\boldsymbol{a}^2 \leftarrow \boldsymbol{h}^1 * \boldsymbol{w}^2 \text{ (valid convolution)} \tag{5}$$
$$\boldsymbol{h}^2 \leftarrow \text{ReLU}(\boldsymbol{a}^2) \tag{6}$$
$$\ldots \tag{7}$$
$$L \leftarrow \ldots \tag{8}$$

where L is the loss.

6.1 What is the dimensionality of $\boldsymbol{a}^2$? Denote it by $|\boldsymbol{a}^2|$.

For valid convolution without flipping, we know that output is defined only where the kernel completely falls in the domain of the input, so we can obtain the dimensionality of $|\boldsymbol{a}^1|$ and then $|\boldsymbol{a}^2|$ becuase ReLU won't change the dimensionality, it just operates on all elements of a vector. So since there's no flipping, output indexes start from 1, and continue until the *last* element of the kernel touches the *last* element of the output. In other words w_{k_1} will touch x_d, so at that specific n we should have $n + k_1 - 1 = d$, So the output's last index is $n = d - k_1 + 1$. We could also achieve this by the formula we saw from the class, since the input size d, the stride is 1, padding is 0, kernel size is k_1, then $|\boldsymbol{a}^1| = \lfloor \frac{d+2\times 0-k_1}{1} \rfloor + 1 = d - k_1 + 1$. Since ReLU doesn't affect the dimensionality, therefore $|\boldsymbol{h}^1| = |\boldsymbol{a}^1|$. We can again use the formula with the second layer to get $|\boldsymbol{a}^2|$. (Again stride is 1 and padding is zero)

$$\begin{aligned} |\boldsymbol{a}^2| &= \lfloor \frac{|\boldsymbol{h}^1| + 2 \times 0 - k_2}{1} \rfloor + 1 = |\boldsymbol{h}^1| - k_2 + 1 = d - k_1 + 1 - k_2 + 1 \\ &= d - k_1 - k_2 + 2 \end{aligned}$$

6.2 Derive $\frac{\partial a_i^2}{\partial h_n^1}$. Answer for all $i \in \{1, \ldots, |\boldsymbol{a}^2|\}$, given a particular n.

$$\boldsymbol{a}_n^2 = (\boldsymbol{h}^1 * \boldsymbol{w}^2)_n = \sum_{j=1}^{k_1} \boldsymbol{h}_{n+j-1}^1 \boldsymbol{w}^2 j$$

Replacing n by i for $\boldsymbol{a}_n^2$, we get :

$$\frac{\partial a_i^2}{\partial h_n^1} = \frac{\partial}{\partial h_n^1} \sum_{j=1}^{k_1} h_{i+j-1}^1 = \mathbb{1}_{n=i+j-1} \times \boldsymbol{w}_j^2$$

This indicator function is zero everywhere except where the condition holds, and for a given pair of n, i that holds when $j = n - i + 1$, so we have that $\frac{\partial a_i^2}{\partial h_n^1} = w_{n-i+1}^2$. But this amounts to zero if the subscript is not in the bounds of the kernel. To summarize :

$$\frac{\partial a_i^2}{\partial h_n^1} = \begin{cases} w_{n-i+1}^2 & \text{if } n-i+1 \leq k_2 \\ w_{n-i+1}^2 & \text{if } n-i+1 \geq 1 \\ 0 & \text{Otherwise} \end{cases} = \begin{cases} w_{n-i+1}^2 & n-k_2+1 \leq i \leq n \\ 0 & \text{Otherwise} \end{cases}$$

6.3 Show that $\nabla_{\boldsymbol{h}^1} L = \nabla_{\boldsymbol{a}^2} L \tilde{*} \text{flip}(\boldsymbol{w}^2)$ (full convolution). Start with

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_i \frac{\partial a_i^2}{\partial h_n^1}$$

We would first plug in the $\frac{\partial a_i^2}{\partial h_n^1}$ from the previous section :

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_i \frac{\partial a_i^2}{\partial h_n^1} = \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_i w_{n-i+1}^2$$

But note that from the previous section, $\frac{\partial a_i^2}{\partial h_n^1} = 0$ if $n - i + 1 \notin [1, \ldots, k_2]$ or equivalently for the partial to be non-zero we need that $i \in [n - k_2 + 1, n]$. So we can change the bounds of the summation :

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_i \frac{\partial a_i^2}{\partial h_n^1} = \sum_{i=n-k_2+1}^{n} (\nabla_{\boldsymbol{a}^2} L)_i w_{n-i+1}^2$$

Next we reorder the sum and define $j = n - i + 1$, then j would run from $1 \to k_2$:

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{j=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n-j+1} w_j^2$$

We recall that $\text{flip}(\boldsymbol{w})_j = w_{k-j+1}$, replacing it :

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{j=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n-j+1} w_j^2 = \sum_{j=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n-j+1} \text{flip}(w^2)_{k_2-j+1}$$

Again changing the variables and letting $m = k_2 - j + 1$ then the sum will again run from $1 \to k_2$. Thus we have :

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{m=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n+m-k_2} \text{flip}(w^2)_m$$

This summation looks exactly the same as the definition of the full convolution : $u \tilde{*} v = \sum_{j=1}^{k} u_{n+j-k} v_j$, if we call $u = \nabla_{\boldsymbol{a}^2} L$ and $v = \text{flip}(w^2)$, So this concludes the derivation and we have that $\nabla_{\boldsymbol{h}^1} L = \nabla_{\boldsymbol{a}^2} L \tilde{*} \text{flip}(\boldsymbol{w}^2)$.

For the following, assume the convolutions in equations (3) and (5) are **full instead of valid**.

6.4 What is the dimensionality of $\boldsymbol{a}^2$? Denote it by $|\boldsymbol{a}^2|$.

For full convolution without flipping, we know that output is defined where even a single element of the kernel touches a single element in the domain of the input, so we can obtain the dimensionality of $|\boldsymbol{a}^1|$ and then $|\boldsymbol{a}^2|$ becuase again ReLU won't change the dimensionality. In the full case, there's $k-1$ padding to each side of the sequence, and then the kernel is moved across this sequence of length $d + 2 \times (k-1)$. First index of the output occurs when w_k touches the first element of the input sequence. The last element occurs when w_1 touches the last element of the input sequence, or equivalently w_k touches x_{d+k-1}, hence the length of the output would be obtained by observing how many steps w_k takes which is from $1 \to d+k-1$. So the output's length is $n = d + k_1 + 1$. We could also achieve this by the formula we saw from the class, since the input size d, the stride is 1, padding is $k-1$, kernel size is k_1, then $|\boldsymbol{a}^1| = \lfloor \frac{d+2\times(k_1-1)-k_1}{1} \rfloor + 1 = d + k_1 - 1$. Since ReLU doesn't affect the dimensionality, therefore $|\boldsymbol{h}^1| = |\boldsymbol{a}^1|$. We can again use the formula with the second layer to get $|\boldsymbol{a}^2|$. (Again stride is 1 and padding is $k_2 - 1$ for each side)

$$\begin{aligned} |\boldsymbol{a}^2| &= \lfloor \frac{|\boldsymbol{h}^1| + 2 \times (k_2 - 1) - k_2}{1} \rfloor + 1 = |\boldsymbol{h}^1| + k_2 - 1 = d + k_1 - 1 + k_2 - 1 \\ &= d + k_1 + k_2 - 2 \end{aligned}$$

6.5 Derive $\frac{\partial a_i^2}{\partial h_n^1}$. Answer for all $i \in \{1, ..., |\boldsymbol{a}^2|\}$, given a particular n.

$$\boldsymbol{a}_n^2 = (\boldsymbol{h}^1 \tilde{*} \boldsymbol{w}^2)_n = \sum_{j=1}^{k_2} \boldsymbol{h}_{n+j-k_2}^1 \boldsymbol{w}^2 j$$

Replacing n by i for $\boldsymbol{a}_n^2$, we get :

$$\frac{\partial a_i^2}{\partial h_n^1} = \frac{\partial}{\partial h_n^1} \sum_{j=1}^{k_2} h_{i+j-k_2}^1 = \mathbb{1}_{n=i+j-k_2} \times \boldsymbol{w}_j^2$$

This indicator function is zero everywhere except where the condition holds, and for a given pair of n, i that holds when $j = n - i + k_2$, so we have that $\frac{\partial a_i^2}{\partial h_n^1} = w_{n-i+k_2}^2$. But this amounts

to zero if the subscript is not in the bounds of the kernel. To summarize :

$$\frac{\partial a_i^2}{\partial h_n^1} = \begin{cases} w^2_{n-i+k_2} & \text{if } n-i+k_2 \leq k_2 \\ w^2_{n-i+k_2} & \text{if } n-i+k_2 \geq 1 \\ 0 & \text{Otherwise} \end{cases} = \begin{cases} w^2_{n-i+k_2} & n \leq i \leq n+k_2-1 \\ 0 & \text{Otherwise} \end{cases}$$

6.6 Show that $\nabla_{\boldsymbol{h}^1} L = \nabla_{\boldsymbol{a}^2} L * \text{flip}(\boldsymbol{w}^2)$ (valid convolution). Start with

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_i \frac{\partial a_i^2}{\partial h_n^1}$$

We would first plug in the $\frac{\partial a_i^2}{\partial h_n^1}$ from the previous section :

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_i \frac{\partial a_i^2}{\partial h_n^1} = \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_i w^2_{n-i+k_2}$$

But note that from the previous section, $\frac{\partial a_i^2}{\partial h_n^1} = 0$ if $n-i+k_2 \notin [1, \ldots, k_2]$ or equivalently for the partial to be non-zero we need that $i \in [n, n+k_2-1]$. So we can change the bounds of the summation :

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_i \frac{\partial a_i^2}{\partial h_n^1} = \sum_{i=n}^{n+k_2-1} (\nabla_{\boldsymbol{a}^2} L)_i w^2_{n-i+k_2}$$

Next we reorder the sum and define $j = n-i+k_2$, then j would run from $1 \to k_2$:

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{j=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n-j+k_2} w_j^2$$

We recall that $\text{flip}(\boldsymbol{w})_j = w_{k-j+1}$, replacing it :

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{j=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n-j+k_2} w_j^2 = \sum_{j=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n-j+k_2} \text{flip}(w^2)_{k_2-j+1}$$

Again changing the variables and letting $m = k_2 - j + 1$ then the sum will again run from $1 \to k_2$. Thus we have :

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{m=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n+m-1} \text{flip}(w^2)_m$$

This summation looks exactly the same as the definition of the valid convolution : $u * v = \sum_{j=1}^{k} u_{n+j-1} v_j$, if we call $u = \nabla_{\boldsymbol{a}^2} L$ and $v = \text{flip}(w^2)$, So this concludes the derivation and we have that $\nabla_{\boldsymbol{h}^1} L = \nabla_{\boldsymbol{a}^2} L \ * \text{flip}(\boldsymbol{w}^2)$.