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.
Answer :
Using the definition of the derivative :

$$\frac{d}{dx}f(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

— If $x > 0$:

$$\frac{d}{dx}g(x) = \lim_{\epsilon \to 0} \frac{g(x+\epsilon) - g(x)}{\epsilon}$$
$$= \lim_{\epsilon \to 0} \frac{\max\{0, x+\epsilon\} - \max\{0, x\}}{\epsilon}$$
$$= \lim_{\epsilon \to 0} \frac{x+\epsilon - x}{\epsilon}$$
$$= \lim_{\epsilon \to 0} \frac{\epsilon}{\epsilon} = 1 = H(x)$$

— If $x < 0$:

$$\frac{d}{dx}g(x) = \lim_{\epsilon \to 0} \frac{g(x+\epsilon) - g(x)}{\epsilon}$$
$$= \lim_{\epsilon \to 0} \frac{\max\{0, x+\epsilon\} - \max\{0, x\}}{\epsilon}$$
$$= \lim_{\epsilon \to 0} \frac{0 - 0}{\epsilon} = 0 = H(x)$$

— If $x = 0$:
Note that technically, the derivative does not exist at x=0. The reason for it being undefined at x=0 is that its left derivative (= 0) and right derivative (= 1) are not equal.

When x=0, there are many possible lines (slopes) we could fit through it. Basically we just choose a slope to use when x=0.
"Typically, we set it either to 0, 1, or 0.5" <source>
So, we can set it equal to $0.5 = H(x)$.

Therefore, the derivative of the rectified linear unit g(x) = max0,x,wherever it exists, is equal to the Heaviside step function.

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

— Alternative 1 :

$$g(x) = x.H(x)$$

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

Therefore, $g(x) = max(0, x) = x.H(x)$.

— Alternative 2 :

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

We know that the derivative of the rectified linear unit g(x) = max0,x,wherever it exists, is equal to the Heaviside step function.$H(x) = g'(x)$. So :

$$\int_a^b H(x)\,\mathrm{d}x = g(b) - g(a) = [g(x)]_a^b$$

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

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

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.
Answer :

$$\lim_{k\to+\infty} \sigma(kx) = \frac{1}{1+e^{-kx}} = 1 = H(x)$$

$$\lim_{k\to-\infty} \sigma(kx) = \frac{1}{1+e^{-kx}} = 0 = H(x)$$

$$\lim_{k\to\infty} \sigma(k * 0) = \frac{1}{1+e^{-k*0}} = \frac{1}{2} = H(0)$$

Therefore, $H(x)$ is well approximated by the sigmoid function for large 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.
Answer :
$S(\boldsymbol{x}+c)_i = \frac{e^{\boldsymbol{x}_i + c}}{\sum_j e^{\boldsymbol{x}_j + c}} = \frac{e^{\boldsymbol{x}_i} e^c}{\sum_j e^{\boldsymbol{x}_j} e^c} = \frac{e^c e^{\boldsymbol{x}_i}}{e^c \sum_j e^{\boldsymbol{x}_j}} = \frac{e^{\boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{x}_j}} = S(\boldsymbol{x})$
Therefore, the softmax function 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 $\boldsymbol{x}$.
Answer :

Using an affine transformation followed by a softmax :

$$\begin{pmatrix} z_0 \\ z_1 \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} \boldsymbol{x} + \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}$$

$$P(C_i|\boldsymbol{x}) = S(z_i) = \frac{e^{z_i}}{e^{z_0} + e^{z_1}},\ i \in \{0, 1\}$$

Transforming it into an equivalent binary classifier that uses a sigmoid instead of the softmax. Choosing class C_0, the probability that we want the sigmoid to output.

$$z = a^T \boldsymbol{x} + b$$

$$P(C_0|\boldsymbol{x}) = \sigma(z) = \frac{1}{1+e^{-z}}$$
$$P(C_1|\boldsymbol{x}) = 1 - \sigma(z)$$

The classifiers are equivalent if the probabilities are the same, so we must impose :

$$\sigma(z) = S(z_0)$$

$$\frac{1}{1+e^{-z}} = \frac{e^{z_0}}{e^{z_0} + e^{z_1}} = \frac{1}{1+e^{z_1 - z_0}}$$

the equality above holds if :

$$a = a_0 - a_1$$
$$b = b_0 - b_1.$$

with $a = a_0 - a_1$ and $b = b_0 - b_1$, we see that we obtain the same probabilities as in the two-class using the sigmoid function.

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 $i \in \{1, ..., K-1\}$.
Answer :
Suppose we have $K-1$ parameters :$w_1, w_2 \ldots w_{K-1}$
The sum of the probability values for each possible output value should sum to 1.

$$P(Y = 1|x; W) = w_1$$
$$P(Y = 2|x;) = w_2$$
$$\vdots$$
$$P(Y = k|x; W) = w_k$$

With all these parameters, we have :

$$\sum_{i=1}^{k} w_i = 1$$

However, the parameters are not independent, one parameter is redundant, because we can get the last parameter by subtracting the sum of the others from 1.

$$w_k = 1 - (w_1 + w_2 + \ldots + w_{k-1})$$

Thus, we have $k-1$ parameters, with $y_i = W^T x$ from w_2 through w_K and define the first one as $w_1 = 0$

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

Answer :
The Jacobian matrix of the softmax function $\frac{\partial S(\boldsymbol{x})}{\partial \boldsymbol{x}}$

$$\frac{\partial S(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial S(\boldsymbol{x})_1}{\partial \boldsymbol{x}} \\ \frac{\partial S(\boldsymbol{x})_2}{\partial \boldsymbol{x}} \\ \vdots \\ \frac{\partial S(\boldsymbol{x})_n}{\partial \boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial S(\boldsymbol{x})_1}{\partial \boldsymbol{x}_1} & \frac{\partial S(\boldsymbol{x})_1}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial S(\boldsymbol{x})_1}{\partial \boldsymbol{x}_n} \\ \frac{\partial S(\boldsymbol{x})_2}{\partial \boldsymbol{x}_1} & \frac{\partial S(\boldsymbol{x})_2}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial S(\boldsymbol{x})_2}{\partial \boldsymbol{x}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial S(\boldsymbol{x})_n}{\partial \boldsymbol{x}_1} & \frac{\partial S(\boldsymbol{x})_n}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial S(\boldsymbol{x})_n}{\partial \boldsymbol{x}_n} \end{bmatrix}$$

Now, we need the partial derivatives of the softmax function :
Using the law of independence :

$$\frac{\partial e^{x_i}}{\partial x_j} = 0$$

The derivative of Exponentials :

$$\frac{\partial e^{x_i}}{\partial x_i} = e_i^x$$

We will consider two cases :

— $i = j$:

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_i} = \frac{\partial}{\partial \boldsymbol{x}_i} \frac{e^{\boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{x}_j}}$$

The derivative of fractions $h(x) = \frac{f(x)}{g(x)}$:

$$\frac{\partial h(x)}{\partial x} = \frac{f'(x)g(x) - g'(x)f'x)}{g(x)^2}$$

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_i} = \frac{\frac{\partial e^{x_i}}{\partial x_i} \sum_j e^{\boldsymbol{x}_j} - \frac{\partial \sum_j e^{\boldsymbol{x}_i}}{\partial x_i} e^{x_i}}{(\sum_j e^{\boldsymbol{x}_j})^2}$$

The derivative of a sum is the sum of derivatives is the sum of derivatives.

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_i} = \frac{e^{x_i} \sum_j e^{\boldsymbol{x}_j} - e^{x_i}.e^{x_i}}{(\sum_j e^{\boldsymbol{x}_j})^2}$$

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_i} = \frac{e^{x_i}}{\sum_j e^{\boldsymbol{x}_j}} \frac{\sum_j e^{\boldsymbol{x}_j} - e^{x_i}}{\sum_j e^{\boldsymbol{x}_j}}$$

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_i} = S(\boldsymbol{x})_i(1 - S(\boldsymbol{x})_i)$$

— $i \neq j$:

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_j} = \frac{\partial}{\partial \boldsymbol{x}_j} \frac{e^{\boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{x}_j}}$$

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_j} = \frac{\frac{\partial e^{x_i}}{\partial x_j} \sum_j e^{\boldsymbol{x}_j} - \frac{\partial \sum_j e^{\boldsymbol{x}_j}}{\partial x_j} e^{x_i}}{(\sum_j e^{\boldsymbol{x}_j})^2}$$

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_j} = \frac{0 \sum_j e^{\boldsymbol{x}_j} - e^{x_i}.e^{x_j}}{(\sum_j e^{\boldsymbol{x}_j})^2}$$

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_j} = \frac{-e^{x_i}}{\sum_j e^{\boldsymbol{x}_j}} \frac{e^{x_j}}{\sum_j e^{\boldsymbol{x}_j}}$$

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_j} = -S(\boldsymbol{x})_i S(\boldsymbol{x})_j$$

Therefore,

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_j} = \begin{cases} S(\boldsymbol{x})_i(1 - S(\boldsymbol{x})_i) & \text{if } i = j \\ -S(\boldsymbol{x})_i S(\boldsymbol{x})_j & \text{if } i \neq j \end{cases} \tag{1}$$

If we use the indicator function, we get :

$$\frac{\partial S(\boldsymbol{x})_i}{\partial \boldsymbol{x}_j} = S(\boldsymbol{x})_i \mathbf{1}_{i=j} - S(\boldsymbol{x})_i S(\boldsymbol{x})_j$$

Return to the Jacobian matrix of the softmax function $\frac{\partial S(\boldsymbol{x})}{\partial \boldsymbol{x}}$

$$= \begin{bmatrix} S(\boldsymbol{x})_1(1 - S(\boldsymbol{x})_1) & S(\boldsymbol{x})_1(0 - S(\boldsymbol{x})_2) & \dots & S(\boldsymbol{x})_1(0 - S(\boldsymbol{x})_n) \\ S(\boldsymbol{x})_2(0 - S(\boldsymbol{x})_1) & S(\boldsymbol{x})_2(1 - S(\boldsymbol{x})_2) & \dots & S(\boldsymbol{x})_2(0 - S(\boldsymbol{x})_n) \\ \vdots & \vdots & \ddots & \vdots \\ S(\boldsymbol{x})_n(0 - S(\boldsymbol{x})_1) & S(\boldsymbol{x})_n(0 - S(\boldsymbol{x})_2) & \dots & S(\boldsymbol{x})_n(1 - S(\boldsymbol{x})_n) \end{bmatrix}$$

$$\frac{\partial S(\boldsymbol{x})}{\partial \boldsymbol{x}} = diag(S(\boldsymbol{x})) - S(\boldsymbol{x})S(\boldsymbol{x})^T$$

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

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.
Answer :
We have :

$$F = h \circ g \circ f \to F(x) = h(g(f(x)))$$

Differentiating the composition of three functions, we need to apply the chain rule more than once.
Applying the chain rule once we obtain :

$$F'(x) = \frac{d}{dx}(h(g(f(x)))) = h'(g(f(x))).\frac{d}{dx}(g(f(x)))$$

Applying the chain rule again, we obtain :

$$F'(x) = h'(g(f(x))).g'(f(x)).f'(x)$$

"the jacobian is just the dot product between the jacobians " <source>
In other words, The Jacobian matrix of F is :

$$J_F(x) = J_h(g(f(x))).J_g(f(x)).J_f(x)$$

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 $F(x)$.
Answer :

Forward mode accumulation would compute (the chain-rule is computed from the "inside out") :

$$J_F(x) = J_h(g(f(x)))\Bigg(J_g(f(x))J_f(x)\Bigg)$$

Pseudo code :

```
$w_0 \leftarrow x$
$w_1 \leftarrow f(x)$
$w_2 \leftarrow g(w_1)$
$F \leftarrow h(w_2)$
$jacob \leftarrow matmul(J_g(w_1), J_f(w_0))$
$J_F \leftarrow matmul(J_h(w_2), jacob)$
return $F$,$J_F$
```

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 `matmul`$(\cdot, \cdot)$ which performs matrix multiplication (no limit on the number of calls). Your pseudo code should also return $F(x)$.
Answer :
Reverse mode accumulation would compute(the chain-rule is computed from the "outside in") :

$$J_F(x) = \Big(J_h(g(f(x)))J_g(f(x))\Big)J_f(x)$$

Pseudo code :

```
$w_0 \leftarrow x$
$w_1 \leftarrow f(x)$
$w_2 \leftarrow g(w_1)$
$F \leftarrow h(w_2)$
$jacob \leftarrow matmul(J_h(w_2), J_g(w_1))$
$J_F \leftarrow matmul(jacob, , J_f(w_0))$
return $F$,$J_F$
```

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 ?
Answer :
Knowing that matrix multiplication has the biggest time complexity in the algorithm for that the time complexity is equal to the time complexity of the matrices multiplications.

Also,the number of scalar multiplications required to multiply two matrices of sizes a×b and b×c is axbxc. So :

- Time complexity of forward mode pseudo code $= O(lmn)+O(lno) = \max\{O(lmn), O(lno)\}$
- Time complexity of reverse mode pseudo code $= O(omn)+O(oml) = \max\{O(omn), O(oml)\}$

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

Answer :

— *Valid convolution* : it means no padding :
$[5, 6, 7, 8] * [3, 0, 1] = [26, 30]$

— *Full convolution* :the maximum padding which does not result in a convolution over just padded elements :
$[5, 6, 7, 8] * [3, 0, 1] = [15, 18, 26, 30, 7, 8]$

— *Same convolution* :here we apply padding so that the output size is the same as the input :
$[5, 6, 7, 8] * [3, 0, 1] = [18, 26, 30, 7]$

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 ?
Answer :
We can compute the output dimension :

— i : input

— k : kernel size

— s : stride

— p : padding

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

Input layer : $3 \times 128 \times 128$

— **First layer :** Convolution layer
input : 128
convolutions : 32
kernel size (k) : 8×8
stride : 2
padding : 3

$$output = \lfloor \frac{128 + 2 * 3 - 8}{2} \rfloor + 1 = 64$$

output : 32 $\times$ 64 $\times$ 64

— **Second layer :** Max pooling layer
input : 64
kernel size (k) : 2 x 2
stride : 2 (no overlapping) padding : 0

$$output = \lfloor \frac{64 + 2 * 0 - 2}{2} \rfloor + 1 = 32$$

output : 32 $\times$ 32 $\times$ 32

— **Third layer :**Convolution layer
input : 32
convolutions : 64
kernel size (k) : 3×3
stride : 1
padding : 1

$$output = \lfloor \frac{32 + 2 * 1 - 3}{1} \rfloor + 1 = 32$$

output : 64 $\times$ 32 $\times$ 32
There is $64 \times 32 \times 32 = 65536$ dimensions (scalars) for this third and last layer.

5.2 Not including the biases, how many parameters are needed for the last layer ?
Answer :
Number of parameters in a CONV layer : $(m * n * d) * k)$ where width m, height n, previous layer's filters d and account for all such filters k in the current layer.

$$Number of parameters = (3 * 3 * 32) * 64 = 18432$$

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

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

$$\text{full :} (\boldsymbol{x} \tilde{*} \boldsymbol{w})_n = \sum_{j=1}^{k} x_{n+j-k} w_j \ , \tag{3}$$

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 :

$$\boldsymbol{a}^1 \leftarrow \boldsymbol{x} * \boldsymbol{w}^1 \text{ (valid convolution)} \tag{4}$$
$$\boldsymbol{h}^1 \leftarrow \text{ReLU}(\boldsymbol{a}^1) \tag{5}$$
$$\boldsymbol{a}^2 \leftarrow \boldsymbol{h}^1 * \boldsymbol{w}^2 \text{ (valid convolution)} \tag{6}$$
$$\boldsymbol{h}^2 \leftarrow \text{ReLU}(\boldsymbol{a}^2) \tag{7}$$
$$\ldots \tag{8}$$
$$L \leftarrow \ldots \tag{9}$$

where L is the loss.

6.1 What is the dimensionality of $\boldsymbol{a}^2$? Denote it by $|\boldsymbol{a}^2|$.
Answer :
The output dimension from a convolution layer with valid padding (no padding) knowing the input dimension = n x n and the kernel size = f x f is equal to :

$$output = (n - f + 1) * (n - f + 1)$$

— We have $\boldsymbol{x} \in \mathbb{R}^d \to \boldsymbol{x}$ dimension $= d * 1$ and $\boldsymbol{w}^1 \in \mathbb{R}^{k_1} \to \boldsymbol{w}^1$ dimension $= k_1 * 1$
So,$\boldsymbol{a}^1$ dimension $= (d - k_1 + 1) * 1$
We have $\boldsymbol{h}^2 = \text{ReLU}(\boldsymbol{a}^2)$, therefore, h^1 dimension $= \boldsymbol{a}^1$ dimension $= (d - k_1 + 1) * 1$

— We follow the same reasoning to calculate the dimensionality of $\boldsymbol{a}^2$: knowing that $\boldsymbol{w}^2 \in \mathbb{R}^{k_2}$ $\to \boldsymbol{w}^2$ dimension $= k_2 * 1$ and h^1 dimension $= (d - k_1 + 1) * 1$
So,$\boldsymbol{a}^2$ dimension $= (d - k_1 + 1 - k_2 + 1) * 1 = (d - k_1 - k_2 + 2) * 1$

$$|\boldsymbol{a}^2| = d - k_1 - k_2 + 2$$

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

$$\boldsymbol{a}^2 = \boldsymbol{h}^1 * \boldsymbol{w}^2$$

$$\boldsymbol{a}_i^2 = (\boldsymbol{h}^1 * \boldsymbol{w}^2)_i = \sum_{j=1}^{k_2} h_{i+j-1}^1 w_j^2$$

$$\frac{\partial a_i^2}{\partial h_n^1} = \frac{\partial}{\partial h_n^1} \sum_{j=1}^{k_2} h_{i+j-1}^1 w_j^2$$

$$\frac{\partial a_i^2}{\partial h_n^1} = \sum_{j=1}^{k_2} \frac{\partial}{\partial h_n^1} h_{i+j-1}^1 w_j^2$$

By expanding the summation we end up observing that the derivative will only be non-zero when $i + j - 1 = n \rightarrow i = n - j + 1 \rightarrow j = n - i + 1$ for $1 \leq j \leq k_2$

$$\frac{\partial a_i^2}{\partial h_n^1} = w_{n-i+1}^2$$

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

Answer :

Starting with :

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

$(\nabla_{\boldsymbol{h}^1} L)_n$ only exists when :

$$\frac{\partial a_i^2}{\partial h_n^1} = w_{n-i+1}^2$$

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

for $1 \leq j \leq k_2$

Changing the variable of the indices $i+j-1 = n \to i = n-j+1 \to j = n-i+1$ for $1 \leq j \leq k_2$, we get :

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

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

$$\nabla_{\boldsymbol{h}^1} L = \nabla_{\boldsymbol{a}^2} L \tilde{*} \text{flip}(\boldsymbol{w}^2)$$

For the following, assume the convolutions in equations (4) and (6) are **full instead of valid**.

6.4 What is the dimensionality of $\boldsymbol{a}^2$? Denote it by $|\boldsymbol{a}^2|$.
Answer :
FULL padding is the maximum padding which does not result in a convolution over just padded elements. For a kernel of size k, this padding is equal to $k-1$.
We can compute the output dimension :

$$out = \lfloor \frac{in + 2p - k}{s} \rfloor + 1 = \lfloor \frac{in + 2(k-1) - k}{s} \rfloor + 1 =$$

— We have $\boldsymbol{x} \in \mathbb{R}^d \rightarrow \boldsymbol{x}$ dimension $= d * 1$ and $\boldsymbol{w}^1 \in \mathbb{R}^{k_1} \rightarrow \boldsymbol{w}^1$ dimension $= k_1 * 1$
So,$\boldsymbol{a}^1$ dimension $= (d + k_1 - 1) * 1$
We have $\boldsymbol{h}^2 = \text{ReLU}(\boldsymbol{a}^2)$, therefore, h^1 dimension $= \boldsymbol{a}^1$ dimension $= (d + k_1 - 1) * 1$

— We follow the same reasoning to calculate the dimensionality of $\boldsymbol{a}^2$: knowing that $\boldsymbol{w}^2 \in \mathbb{R}^{k_2}$ $\rightarrow \boldsymbol{w}^2$ dimension $= k_2 * 1$ and h^1 dimension $= (d + k_1 - 1) * 1$
So,$\boldsymbol{a}^2$ dimension $= (d + k_1 + k_2 - 2) * 1$

$$|\boldsymbol{a}^2| = d + k_1 + k_2 - 2$$

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.
Answer :
We have :

$$\boldsymbol{a}^2 = \boldsymbol{h}^1 * \boldsymbol{w}^2$$

$$\boldsymbol{a}_i^2 = (\boldsymbol{x} \tilde{*} \boldsymbol{w})_i = \sum_{j=1}^{k} {}^2 h_{i+j-k^2}^1 w_j^2$$

$$\frac{\partial a_i^2}{\partial h_n^1} = \frac{\partial}{\partial h_n^1} \sum_{j=1}^{k_2} h_{i+j-k^2}^1 w_j^2$$

$$\frac{\partial a_i^2}{\partial h_n^1} = \sum_{j=1}^{k_2} \frac{\partial}{\partial h_n^1} h_{i+j-k^2}^1 w_j^2$$

By expanding the summation we end up observing that the derivative will only be non-zero when $i + j - k^2 = n \rightarrow i = n - j + k^2 \rightarrow j = n - i + k^2$ for $1 \leq j \leq k_2$

$$\frac{\partial a_i^2}{\partial h_n^1} = w_{n-i+k^2}^2$$

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

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

$(\nabla_{\boldsymbol{h}^1} L)_n$ only exists when :

$$\frac{\partial a_i^2}{\partial h_n^1} = w_{n-i+k^2}^2$$

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

for $1 \leq j \leq k_2$
Changing the variable of the indices $i + j - k^2 = n \rightarrow i = n - j + k^2 \rightarrow j = n - i + k^2$ for $1 \leq j \leq k_2$, we get :

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

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

$$\nabla_{\boldsymbol{h}^1} L = \nabla_{\boldsymbol{a}^2} L \tilde{*} \text{ flip}(\boldsymbol{w}^2)$$