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

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. In the following, we consider $|x| > |\epsilon|$. Evaluating two one-sided derivatives, we have :

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x} g(x)_{x>0} &= \lim_{\substack{\epsilon \to 0 \\ x>0}} \frac{g(x+\epsilon) - g(x)}{\epsilon} \\ &= \lim_{\substack{\epsilon \to 0 \\ x>0}} \frac{\max\{0, x+\epsilon\} - \max\{0, x\}}{\epsilon} \\ &= \lim_{\substack{\epsilon \to 0 \\ x>0}} \frac{x+\epsilon-x}{\epsilon} = \lim_{\substack{\epsilon \to 0 \\ x>0}} \frac{\epsilon}{\epsilon} = 1 \end{aligned}$$

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x} g(x)_{x<0} &= \lim_{\substack{\epsilon \to 0 \\ x<0}} \frac{g(x+\epsilon) - g(x)}{\epsilon} \\ &= \lim_{\substack{\epsilon \to 0 \\ x<0}} \frac{\max\{0, x+\epsilon\} - \max\{0, x\}}{\epsilon} \\ &= \lim_{\substack{\epsilon \to 0 \\ x<0}} \frac{0-0}{\epsilon} = 0 \end{aligned}$$

Since, the left and right derivative at $x = 0$ of the ReLU are not equal, it is not differentiable at this point. We cant then say that

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

and that it is undefined at $x = 0$.

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

Answer. ReLU can also be expressed as

$$\begin{aligned} g(x) &= xH(x) \\ g(x) &= \int_{-\infty}^{x} H(x)dx \end{aligned}$$

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. For large k and $x > 0$,

$$\begin{aligned}\lim_{k\to\infty} \sigma(x > 0, k) &= \lim_{k\to\infty} \frac{1}{1 + e^{-k|x|}} \\ &= \frac{1}{1} = 1\end{aligned}$$

We examine the case where $x < 0$:

$$\begin{aligned}\lim_{k\to\infty} \sigma(x < 0, k) &= \lim_{k\to\infty} \frac{1}{1 + e^{k|x|}} \\ &= \frac{1}{\infty} = 0\end{aligned}$$

Finally, for $x = 0$,

$$\begin{aligned}\sigma(x = 0, k) &= \frac{1}{1 + e^{0}} \\ &= \frac{1}{2}\end{aligned}$$

which is the convention for the midpoint of $H(x)$. We have shown that the Heaviside function is well approximated by the parametrized sigmoid with 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. Defining $e^{\boldsymbol{x}}$ as a vector of the exponential of each element in $\boldsymbol{x}$,

$$\begin{aligned}S(\boldsymbol{x} + c) &= \frac{e^{\boldsymbol{x}+c}}{\sum_j e^{x_j+c}} \\ &= \frac{e^c e^{\boldsymbol{x}}}{e^c \sum_j e^{x_j}} \\ &= \frac{e^{\boldsymbol{x}}}{\sum_j e^{x_j}} = S(\boldsymbol{x})\end{aligned}$$

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. With binary classification, a model need only output a single scalar score for one of the classes with which to calculate a class probability, since the probability of the second class

is determined exactly by the one of the first. A sigmoid applied to the output of a model $z(\boldsymbol{x})$ represents the probability of $\boldsymbol{x}$ belonging to the "positive" class, which we call A :

$$p(A) = \sigma(z(\boldsymbol{x})) = \frac{1}{1 + e^{-z(\boldsymbol{x})}} = \frac{e^{z(\boldsymbol{x})}}{1 + e^{z(\boldsymbol{x})}}$$

The complementary probability that $\boldsymbol{x}$ belongs to the class *not* A is then simply

$$p(not\ A) = 1 - \sigma(z(\boldsymbol{x})) = \frac{e^{-z(\boldsymbol{x})}}{1 + e^{-z(\boldsymbol{x})}} = \frac{1}{1 + e^{z(\boldsymbol{x})}}$$

Alternatively, suppose that the output of the model given a 2-dimensional vector $\boldsymbol{x}$ was another 2-dimensional vector, with each entry specifying the score of classes A and *not* A in such a way where the first would be $z(\boldsymbol{x})$ and the second 0. Applying the softmax function to such a vector would result in

$$\begin{aligned} S(\boldsymbol{x}) &= \left[\frac{e^{z(\boldsymbol{x})}}{1 + e^{z(\boldsymbol{x})}}, \frac{1}{1 + e^{z(\boldsymbol{x})}}\right]^\top \\ &= \left[\sigma(z), 1 - \sigma(z)\right]^\top . \end{aligned}$$

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. Let the scalar function for a class k be a linear predictor function that outputs a score for this class from K-dimensional vector $\boldsymbol{x}$ with a vector of weights $\boldsymbol{\beta}_k$:

$$y_k = \boldsymbol{\beta}_k \cdot \boldsymbol{x}.$$

The probability that $\boldsymbol{x}$ belongs to class k is given by $S(\boldsymbol{x})_k$:

$$\Pr(c = k) = S(\boldsymbol{x})_k = \frac{e^{\boldsymbol{\beta}_k \cdot \boldsymbol{x}}}{\sum_j e^{\boldsymbol{\beta}_j \cdot \boldsymbol{x}}}.$$

Since the sum of the probabilities across every class must be 1 by definition, a vector $\boldsymbol{\beta}_k$ is completely determined once the rest are known. Recalling the result from question 2.1 where it was shown that softmax is translation invariant, a constant vector $\boldsymbol{C}$ can also be added to all coefficient vectors and still leave the equation unchanged :

$$\frac{e^{(\boldsymbol{\beta}_k + \boldsymbol{C}) \cdot \boldsymbol{x}}}{\sum_j e^{(\boldsymbol{\beta}_j + \boldsymbol{C}) \cdot \boldsymbol{x}}} = \frac{e^{\boldsymbol{\beta}_k \cdot \boldsymbol{x}}}{\sum_j e^{\boldsymbol{\beta}_j \cdot \boldsymbol{x}}}$$

It is then convenient to choose $\boldsymbol{C} = -\boldsymbol{\beta}_0$, for example. We are then left with the set of scalar functions

$$\begin{aligned} y_0 &= (\boldsymbol{\beta}_0 - \boldsymbol{\beta}_0) \cdot \boldsymbol{x} = 0 \\ y_1 &= (\boldsymbol{\beta}_1 - \boldsymbol{\beta}_0) \cdot \boldsymbol{x} = \boldsymbol{\beta}'_1 \cdot \boldsymbol{x} \\ &\vdots \\ y_{K-1} &= (\boldsymbol{\beta}_{K-1} - \boldsymbol{\beta}_0) \cdot \boldsymbol{x} = \boldsymbol{\beta}'_{K-1} \cdot \boldsymbol{x}. \end{aligned}$$

The number of parameters required to represent the softmax has then been reduced to $K-1$:

$$S(\boldsymbol{x}) = S([0, y_1, y_2, \ldots, y_{K-1}]^\top)$$

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. We evaluate the derivative of the ith component of $S(\boldsymbol{x})$ with respect to component x_j :

$$\begin{aligned}
\frac{\partial}{\partial x_j} S(\boldsymbol{x})_i &= \frac{\partial}{\partial x_j}\left(\frac{e^{x_i}}{\sum_k e^{x_k}}\right) \\
&= \frac{1}{\sum_k e^{x_k}} \frac{\partial}{\partial x_j} e^{x_i} + e^{x_i}\left(\frac{1}{\sum_k e^{x_k}}\right) \\
&= \frac{e^{x_i}\delta_{ij}}{\sum_k e^{x_k}} - \frac{e^{x_i}}{\left(\sum_k e^{x_k}\right)^2} \sum_k \frac{\partial}{\partial x_j} e^{x_k} \\
&= \delta_{ij} S(\boldsymbol{x})_j - \frac{e^{x_i} e^{x_j}}{\left(\sum_k e^{x_k}\right)^2} \\
&= \delta_{ij} S(\boldsymbol{x})_j - S(\boldsymbol{x})_i S(\boldsymbol{x})_j .
\end{aligned}$$

These two terms can be interpreted as follows : the first only survives for $i = j$, and the second represents the product for every combination of i and j. Extrapolating back to matrix notation, this corresponds to

$$J_{\text{softmax}}(\boldsymbol{x}) = \mathbf{Diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^\top$$

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. Using the chain rule, the Jacobian matrix of F evaluated at $x \in \mathbb{R}^\ell$ is

$$\begin{aligned}
J_F(x) &= J_{h \circ g \circ f}(x) \\
&= J_h((g \circ f)(x)))J_{g \circ f}(x) \\
&= J_h(g(f(x)))J_g(f(x))J_f(x)
\end{aligned}$$

Or, in matrix form,

$$\begin{aligned}
J_F(x) &= \begin{bmatrix} \frac{\partial h}{\partial g_1} & \cdots & \frac{\partial h}{\partial g_n} \end{bmatrix} \begin{bmatrix} \frac{\partial g}{\partial f_1} & \cdots & \frac{\partial g}{\partial f_m} \end{bmatrix} \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_\ell} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial h_1}{\partial g_1} & \cdots & \frac{\partial h_1}{\partial g_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_o}{\partial g_1} & \cdots & \frac{\partial h_o}{\partial g_n} \end{bmatrix} \begin{bmatrix} \frac{\partial g_1}{\partial f_1} & \cdots & \frac{\partial g_1}{\partial f_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial f_1} & \cdots & \frac{\partial g_n}{\partial f_m} \end{bmatrix} \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_\ell} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_\ell} \end{bmatrix}
\end{aligned}$$

From left to right, the Jacobian matrices are evaluated at $g(f(x)) \in \mathbb{R}^n$, $f(x) \in \mathbb{R}^m$ and $x \in \mathbb{R}^\ell$ respectively.

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 $\texttt{matmul}(\cdot, \cdot)$ which performs matrix multiplication (no limit on the number of calls). Your pseudo code should also return $F(x)$.

Answer. In the algorithm below, the outputs of functions $f,\ g,\ h$ through the forward pass are saved in vectors $\mathbf{v}_f,\ \mathbf{v}_g,\ F$.

Algorithm 1 Forward mode accumulation

Require: $\boldsymbol{x} \in \mathbb{R}^\ell$, the input
1: $\mathbf{v}_f \leftarrow f(\boldsymbol{x})$
2: $\mathbf{v}_g \leftarrow g(\mathbf{v}_f)$
3: $F \leftarrow h(\mathbf{v}_g)$
4: $\nabla_{\boldsymbol{x}} \mathbf{v}_f \leftarrow J_f(\boldsymbol{x})$
5: $\nabla_{\boldsymbol{x}} \mathbf{v}_g \leftarrow \texttt{matmul}(J_g(\mathbf{v}_f), \nabla_{\boldsymbol{x}} \mathbf{v}_f)$
6: $J_F(\boldsymbol{x}) \leftarrow \texttt{matmul}(J_h(\mathbf{v}_g), \nabla_{\boldsymbol{x}} \mathbf{v}_g)$
7: **return** $J_F(\boldsymbol{x})$ and $F(\boldsymbol{x})$

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 $F(x)$.

Answer. The same state variables as in question 3.2 were employed here.

Algorithm 2 Reverse mode accumulation

Require: $\boldsymbol{x} \in \mathbb{R}^\ell$, the input
1: $\mathbf{v}_f \leftarrow f(\boldsymbol{x})$
2: $\mathbf{v}_g \leftarrow g(\mathbf{v}_f)$
3: $F \leftarrow h(\mathbf{v}_g)$
4: $\nabla_{\mathbf{v}_g} F \leftarrow J_h(\mathbf{v}_g)$
5: $\nabla_{\mathbf{v}_f} F \leftarrow \texttt{matmul}(\nabla_{\mathbf{v}_g} F, J_g(\mathbf{v}_f))$
6: $J_F(\boldsymbol{x}) \leftarrow \texttt{matmul}(\nabla_{\mathbf{v}_f} F, J_f(\boldsymbol{x}))$
7: **return** $J_F(\boldsymbol{x})$ and $F(\boldsymbol{x})$

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. Consider two matrices $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{m \times p}$, and suppose that we use an algorithm for which the time complexity of the matrix multiplication AB is $O(nmp)$. Going through each step of the forward accumulation algorithm, the evaluations of $\mathbf{v}_f,\ \mathbf{v}_g,\ F,\ J_f,\ J_g$ and J_h have a time complexity of at least $O(2\ell m + 2mn + 2no) = O(\ell m + mn + no)$. At step 5, we have the

product of $J_g(\mathbf{v}_f) \in \mathbb{R}^{n\times m}$ and $\nabla_{\boldsymbol{x}}\mathbf{v}_f \in \mathbb{R}^{m\times \ell}$, of complexity $O(nm\ell)$, followed by the product of $J_h(\mathbf{v}_g) \in \mathbb{R}^{o\times n}$ and $\nabla_{\boldsymbol{x}}\mathbf{v}_g \in \mathbb{R}^{n\times \ell}$ at step 6 of complexity $O(on\ell)$. The overall time complexity of the forward accumulation algorithm is then $O(\ell m + mn + no + nm\ell + on\ell) = O(nm\ell + on\ell)$. For the reverse accumulation algorithm, the matrix multiplications differ ; at step 5, the product of $\nabla_{\mathbf{v}_g}F \in \mathbb{R}^{o\times n}$ and $J_g(\mathbf{v}_f) \in \mathbb{R}^{n\times m}$ is of complexity $O(onm)$, while that of $\nabla_{\mathbf{v}_f}F \in \mathbb{R}^{o\times m}$ and $J_f(\boldsymbol{x}) \in \mathbb{R}^{m\times \ell}$ at step six is of $O(om\ell)$. The overall complexity for the reverse pass is then $O(\ell m + mn + no + onm + om\ell) = O(onm + om\ell)$.

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

Answer. We define the input $I = [5, 6, 7, 8]$ and the kernel $K = [3, 0, 1]$. The convolution operation, in the machine learning formalism, would then be computed by flipping the kernel from left to right and taking its dot product with overlapping parts of the input, while sliding it along by increments. The kernel is flipped to ensure the commutativity of the operation. A full convolution is one in which for a kernel of dimension k, we apply zero-padding to I so that every element is visited k times. With $k = 3$ in this case, this implies a zero-padding of $2 : I' = [0, 0, 5, 6, 7, 8, 0, 0]$. The result of such a convolution would be :

$$\begin{aligned}
(I * K)_1 &= 1 \cdot 0 + 0 \cdot 0 + 3 \cdot 5 = 15 \\
(I * K)_2 &= 1 \cdot 0 + 0 \cdot 5 + 3 \cdot 6 = 18 \\
(I * K)_3 &= 1 \cdot 5 + 0 \cdot 6 + 3 \cdot 7 = 26 \\
(I * K)_4 &= 1 \cdot 6 + 0 \cdot 7 + 3 \cdot 8 = 30 \\
(I * K)_5 &= 1 \cdot 7 + 0 \cdot 8 + 3 \cdot 0 = 7 \\
(I * K)_6 &= 1 \cdot 8 + 0 \cdot 0 + 3 \cdot 0 = 8 \\
\Rightarrow (I * K) &= \big[15, 18, 26, 30, 7, 8\big] .
\end{aligned}$$

In a valid convolution, the kernel must fit inside the input at all times. Therefore, there is no zero-padding, and we use I as is :

$$\begin{aligned}
(I * K)_1 &= 1 \cdot 5 + 0 \cdot 6 + 3 \cdot 7 = 26 \\
(I * K)_2 &= 1 \cdot 6 + 0 \cdot 7 + 3 \cdot 8 = 30 \\
\Rightarrow (I * K) &= \big[26, 30\big] .
\end{aligned}$$

In a same convolution, the input is zero-padded to ensure that the output is of the same dimension. We would have $I' = [0, 5, 6, 7, 8, 0]$, and the convolution would be :

$$\begin{aligned}
(I * K)_1 &= 1 \cdot 0 + 0 \cdot 5 + 3 \cdot 6 = 18 \\
(I * K)_2 &= 1 \cdot 5 + 0 \cdot 6 + 3 \cdot 7 = 26 \\
(I * K)_3 &= 1 \cdot 6 + 0 \cdot 7 + 3 \cdot 8 = 30 \\
(I * K)_4 &= 1 \cdot 7 + 0 \cdot 8 + 3 \cdot 0 = 7 \\
\Rightarrow (I * K) &= \big[18, 26, 30, 7\big] .
\end{aligned}$$

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 ?

Answer. For a square input/activation feature map and kernel, the output size of a convolution will be

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

where i is the width of the input, p the zero-padding of the kernel, k its width and s its stride. Note that a kernel always extends through the third dimension to cover all the channels. At the first layer, we have $i = 128$, $k = 8$, $s = 2$ and $p = 3$, giving an output width of 64. Stacking the output of each kernel convolution at that layer, the output tensor is of dimensions $32 \times 64 \times 64$. At the second layer, a 2×2 max-pool with no overlapping corresponds to $k = 2$, $p = 0$ and $s = 2$. With $i = 64$, we have an output width of 32. Pooling conserves the third dimension of the input, so the output of the second layer is of dimension $32 \times 32 \times 32$. At the third layer, we have $i = 32$, $k = 3$, $s = 1$ and $p = 1$, resulting in a output width of 32. From the 64 kernels at that layer, the dimension of the third layer is $64 \times 32 \times 32$. The dimensionality of such an output is then 65 536 scalars.

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

Answer. The number of parameters of a given convolution layer with square kernels of width k, disregarding the biases, is $n_{par} = k^2 pc$, where p is the number of filters at the previous layer and c is the number of filters at the current layer. The third layer has $k = 3$, $c = 64$ and $p = 32$ from the second layer. Therefore, the last layer requires 18 432 parameters.

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 :

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

where L is the loss.

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

Answer. In a valid convolution, we have have no zero-padding. From the definition above, we infer a stride of $s = 1$. With expression (1), we compute the dimension of the output of the first layer to be $|\boldsymbol{a}^1| = \left\lfloor \frac{d-k_1}{1} \right\rfloor + 1 = d - k_1 + 1$. The ReLU activation is applied element wise, and therefore leaves the dimension unchanged. The output dimension of the second valid convolution is then $|\boldsymbol{a}^2| = \left\lfloor \frac{d-k_1+1-k_2}{1} \right\rfloor + 1 = d - k_1 - k_2 + 2$.

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. The i^{th} component of vector $\boldsymbol{a}^2$ is obtained by the valid convolution

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

The derivative is then

$$\begin{aligned}\frac{\partial a_i^2}{\partial h_n^1} &= \frac{\partial}{\partial h_n^1} \sum_{j=1}^{k_2} h_{i+j-1}^1 w_j^2 \\ &= \sum_{j=1}^{k_2} \frac{\partial h_{i+j-1}^1}{\partial h_n^1} w_j^2 \\ &= \delta_{n,i+j-1} w_j^2\end{aligned}$$

which is nonzero for $n = i + j - 1 \rightarrow j = n - i + 1$. We then have

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

For $i \in \{1, ..., |\boldsymbol{a}^2|\}$, and with the result from 6.1, this derivative will be :

$$\begin{aligned}\frac{\partial a_1^2}{\partial h_n^1} &= w_n^2 \\ \frac{\partial a_2^2}{\partial h_n^1} &= w_{n-1}^2 \\ &\vdots \\ \frac{\partial}{\partial h_n^1} a_{d-k_1-k_2+2}^2 &= w_{n-d+k_1+k_2+3}^2\end{aligned}$$

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

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

Answer. We begin by developing the expression in the question above, using the result from 6.2 that $\frac{\partial a_i^2}{\partial h_n^1}$ is nonzero for $i = n - j + 1$:

$$\begin{aligned}(\nabla_{\boldsymbol{h}^1} L)_n &= \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_i \frac{\partial a_i^2}{\partial h_n^1} \\ &= \sum_{i=1}^{|\boldsymbol{a}^2|} (\nabla_{\boldsymbol{a}^2} L)_{n-j+1} w_j^2\end{aligned}$$

For a given n in the input layer h_n^1, the output layer a_i^2 of a convolution will have derivatives w.r.t. h_n^1 for $i \in \{n - k_2 + 1, \ldots, n\}$, corresponding to an index $j \in \{1, \ldots, k_2\}$. We can then disregard the terms of the sum outside this range, and sum over j :

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{j=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n-j+1} w_j^2 \tag{10}$$

We introduce an index j' so that $n - j + 1 = n + j' - k_2 \longrightarrow j' = k_2 - j + 1$, $j = k_2 - j' + 1$. The expression now sums over decreasing j' :

$$\begin{aligned}(\nabla_{\boldsymbol{h}^1} L)_n &= \sum_{j'=k_2}^{1} (\nabla_{\boldsymbol{a}^2} L)_{n+j'-k_2} w_{k_2-j'+1}^2 \\ &= \sum_{j'=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n+j'-k_2} w_{k_2-j'+1}^2 \\ &= \left(\nabla_{\boldsymbol{a}^2} L \,\tilde{*}\, \text{flip}(\boldsymbol{w}^2)\right)_n\end{aligned}$$

where the order of summation has been reversed, unaffecting the result.

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. In a full convolution, we have have zero-padding of $k - 1$. From expression (3), we infer a stride of $s = 1$. With expression (1), we compute the dimension of the output of the first layer to be $|\boldsymbol{a}^1| = \left\lfloor \frac{d+2(k_1-1)-k_1}{1} \right\rfloor + 1 = d + k_1 - 1$. The ReLU activation is applied element wise, and therefore leaves the dimension unchanged. The output dimension of the second valid convolution is then $|\boldsymbol{a}^2| = \left\lfloor \frac{d+k_1-1+2(k_2-1)-k_2}{1} \right\rfloor + 1 = d + k_1 + k_2 - 2$.

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. The i^{th} component of vector $\boldsymbol{a}^2$ is obtained by the full convolution

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

The derivative is then

$$\begin{aligned}\frac{\partial a_i^2}{\partial h_n^1} &= \frac{\partial}{\partial h_n^1} \sum_{j=1}^{k_2} h_{i+j-k_2}^1 w_j^2 \\ &= \sum_{j=1}^{k_2} \frac{\partial h_{i+j-k_2}^1}{\partial h_n^1} w_j^2 \\ &= \delta_{n,i+j-k_2} w_j^2\end{aligned}$$

which is nonzero for $n = i + j - k_2 \rightarrow j = n - i + k_2$. We then have

$$\frac{\partial a_i^2}{\partial h_n^1} = w_{n-i+k_2}^2.$$

For $i \in \{1, ..., |\boldsymbol{a}^2|\}$, and with the result from 6.4, this derivative will be :

$$\begin{aligned}\frac{\partial a_1^2}{\partial h_n^1} &= w_{n-1+k_2}^2 \\ \frac{\partial a_2^2}{\partial h_n^1} &= w_{n-2}^2 \\ &\vdots \\ \frac{\partial}{\partial h_n^1} a_{d+k_1+k_2-2}^2 &= w_{n-d-k_1+2}^2\end{aligned}$$

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

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

Answer. We begin by developing the expression in the question above, using the result from 6.5 that $\frac{\partial a_i^2}{\partial h_n^1}$ is nonzero for $i = n - j + k_2$:

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

For a given n in the input layer h^1_n, the output layer a^2_i of a convolution will have derivatives w.r.t. h^1_n for $i \in \{n - k_2 + 1, \ldots, n\}$, corresponding to an index $j \in \{1, \ldots, k_2\}$. We can then disregard the terms of the sum outside this range, and sum over j :

$$(\nabla_{\boldsymbol{h}^1} L)_n = \sum_{j=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n-j+k_2} w^2_j \tag{11}$$

We introduce an index j' so that $n - j + k_2 = n + j' - 1 \longrightarrow j' = k_2 - j + 1$, $j = k_2 - j' + 1$. The expression now sums over decreasing j' :

$$\begin{aligned}
(\nabla_{\boldsymbol{h}^1} L)_n &= \sum_{j'=k_2}^{1} (\nabla_{\boldsymbol{a}^2} L)_{n+j'-1} w^2_{k_2-j'+1} \\
&= \sum_{j'=1}^{k_2} (\nabla_{\boldsymbol{a}^2} L)_{n+j'-1} w^2_{k_2-j'+1} \\
&= \left(\nabla_{\boldsymbol{a}^2} L \ * \ \text{flip}(\boldsymbol{w}^2)\right)_n
\end{aligned}$$

where the order of summation has been reversed, unaffecting the result.