Homework 2 – Theory

Rishabh Patil
CSCI-GA 2572 Deep Learning (Fall 2025)

October 6, 2025

1.1 Convolutional Neural Networks

(a) Output dimension with given parameters (1 pt)

We are given an input image of size 21×12, a kernel of size 4×5, stride $S = 4$, and no padding.

To understand how many times the kernel fits, imagine sliding the kernel across the input. Along the height:

- The first placement covers rows 1 to 4.
- After moving by stride 4, it covers rows 5 to 8.
- Continuing this process, the last valid placement is rows 17 to 20.
- If we try one more step, the kernel would need rows 21 to 24, which do not exist.

Thus there are 5 valid placements in the vertical direction.

Along the width:

- The first placement covers columns 1 to 5.
- The next placement covers columns 5 to 9.
- Trying another step would require columns 9 to 13, but the image only has 12 columns.

Thus there are 2 valid placements horizontally.

Formally, if the input length is L, kernel size K, and stride S, then after t steps the kernel spans

$$[tS,\ tS + K - 1].$$

For this to remain valid, we require

$$tS + (K - 1) \leq L - 1.$$

This inequality gives

$$t \leq \frac{L - K}{S}.$$

The maximum integer t allowed is $\lfloor \frac{L-K}{S} \rfloor$, and since we also include $t = 0$, the number of valid placements is

$$L_{\text{out}} = \left\lfloor \frac{L - K}{S} \right\rfloor + 1.$$

Applying to the given image:

$$H_{\text{out}} = \left\lfloor \frac{21 - 4}{4} \right\rfloor + 1 = 5, \qquad W_{\text{out}} = \left\lfloor \frac{12 - 5}{4} \right\rfloor + 1 = 2.$$

$$\boxed{\text{Output dimension } = 5 \times 2}$$

—

(b) General case (2 pts)

Let the input be $C \times H \times W$ with C channels, kernel size $K \times K$, padding P, stride S, dilation D, and F filters.

Step 1: Effective kernel size. Dilation means that the kernel does not cover K consecutive elements, but instead skips $D-1$ elements between taps. Thus the last index touched by a kernel of size K with dilation D is

$$n + (K-1)D,$$

and the first index touched is

$$n.$$

Therefore, the number of indices covered (the effective kernel size) is

$$\big(n + (K-1)D\big) - n + 1 = (K-1)D + 1.$$

Step 2: Counting placements. As in part (a), the kernel starting at tS extends to $tS + (K-1)D$. For the kernel to fit, we require

$$tS + (K-1)D \le (L + 2P) - 1,$$

where L is the input length in one dimension and $2P$ accounts for padding. This gives the number of valid placements:

$$L_{\text{out}} = \left\lfloor \frac{L + 2P - \big((K-1)D + 1\big)}{S} \right\rfloor + 1.$$

Step 3: Apply to 2D. For height H and width W, the output dimensions are

$$H_{\text{out}} = \left\lfloor \frac{H + 2P - \big((K-1)D + 1\big)}{S} \right\rfloor + 1, \qquad W_{\text{out}} = \left\lfloor \frac{W + 2P - \big((K-1)D + 1\big)}{S} \right\rfloor + 1.$$

Step 4: Final output dimension. Since each of the F filters produces one output channel, the final shape is

$$\boxed{F \times H_{\text{out}} \times W_{\text{out}}}.$$

—

(c) One-Dimensional Convolutions (12 pts)

We are given a one-dimensional convolution with:

- Input $x[n] \in \mathbb{R}^5$ of length 7, i.e. $x \in \mathbb{R}^{5\times 7}$.
- A convolutional layer f_W with one filter, kernel size $K = 3$, stride $S = 2$, no padding or dilation.
- Weights $W \in \mathbb{R}^{1\times 5\times 3}$, with no bias or non-linearity.

The convolution is implemented as cross-correlation:

$$s[i] = \sum_{c=1}^{5} \sum_{m=0}^{2} W[0, c, m]\, x[c,\, iS + m], \qquad i \in \{0, 1, 2\}.$$

—

(i) Dimension and Expression of $f_W(x)$ The output length for a 1-D convolution without padding or dilation is

$$L_{\text{out}} = \left\lfloor \frac{L_{\text{in}} - K}{S} \right\rfloor + 1.$$

Substituting $L_{\text{in}} = 7$, $K = 3$, $S = 2$:

$$L_{\text{out}} = \left\lfloor \frac{7-3}{2} \right\rfloor + 1 = \lfloor 2 \rfloor + 1 = 3.$$

Thus, there are 3 valid output positions: $i \in \{0, 1, 2\}$. Because there is only one filter, the output tensor has shape

$$\boxed{f_W(x) \in \mathbb{R}^{1\times 3}}.$$

At each output position i, the kernel window begins at input index iS and spans offsets $m \in \{0, 1, 2\}$. Hence, the explicit expression is:

$$\boxed{f_W(x)[0,i] = \sum_{c=1}^{5} \sum_{m=0}^{2} W[0,c,m]\, x[c,\, 2i+m], \quad i \in \{0,1,2\},\ c \in \{1,\ldots,5\},\ m \in \{0,1,2\}.}$$

—

(ii) Derivative $\dfrac{\partial \boldsymbol{f_W(x)}}{\partial \boldsymbol{W}}$

$$f_W(x)[0,i] = \sum_{c=1}^{5} \sum_{m=0}^{2} W[0,c,m]\, x[c, 2i+m],$$

we differentiate with respect to a particular $W[0, c_0, m_0]$:

$$\begin{aligned} \frac{\partial \boldsymbol{f_W(x)[0,i]}}{\partial \boldsymbol{W[0,c_0,m_0]}} &= \sum_{c=1}^{5} \sum_{m=0}^{2} \frac{\partial}{\partial \boldsymbol{W[0,c_0,m_0]}} \big(W[0,c,m]\, x[c, 2i+m]\big) \\ &= \sum_{c=1}^{5} \sum_{m=0}^{2} \delta_{c,c_0}\, \delta_{m,m_0}\, x[c, 2i+m] \end{aligned}$$

We use the **Kronecker delta** to express the derivative of one tensor element with respect to another:

$$\frac{\partial W[0,c,m]}{\partial W[0,c_0,m_0]} = \delta_{c,c_0}\, \delta_{m,m_0},$$

where the Kronecker delta is defined as

$$\delta_{p,q} = \begin{cases} 1, & \text{if } p = q, \\ 0, & \text{otherwise.} \end{cases}$$

Substituting this into our earlier expression:

$$\frac{\partial f_W(x)[0,i]}{\partial W[0,c_0,m_0]} = \sum_{c=1}^{5} \sum_{m=0}^{2} \delta_{c,c_0}\, \delta_{m,m_0}\, x[c, 2i+m].$$

All terms vanish except when $c = c_0$ and $m = m_0$, leaving:

$$\boxed{\frac{\partial f_W(x)[0,i]}{\partial W[0,c_0,m_0]} = x[c_0,\, 2i+m_0].}$$

Thus,

$$\boxed{\left(\frac{\partial \boldsymbol{f_W(x)}}{\partial \boldsymbol{W}}\right)[0,i,0,c,m] = x[c,\, 2i+m], \quad i \in \{0,1,2\},\ c \in \{1,\ldots,5\},\ m \in \{0,1,2\}.}$$

The derivative tensor has shape

$$\boxed{\frac{\partial \boldsymbol{f_W(x)}}{\partial \boldsymbol{W}} \in \mathbb{R}^{1\times 3\times 1\times 5\times 3}.}$$

—

(iii) Derivative $\frac{\partial f_W(x)}{\partial x}$

$$f_W(x)[0,i] = \sum_{c=1}^{5} \sum_{m=0}^{2} W[0,c,m]\, x[c, 2i+m],$$

and differentiating with respect to $x[c_0, n]$:

$$\begin{aligned}
\frac{\partial \boldsymbol{f_W(x)[0,i]}}{\partial \boldsymbol{x[c_0,n]}} &= \sum_{c=1}^{5} \sum_{m=0}^{2} W[0,c,m]\, \frac{\partial \boldsymbol{x[c,2i+m]}}{\partial \boldsymbol{x[c_0,n]}} \\
&= \sum_{c=1}^{5} \sum_{m=0}^{2} W[0,c,m]\, \delta_{c,c_0}\, \delta_{2i+m,n} \\
&= \sum_{m=0}^{2} W[0,c_0,m]\, \delta_{2i+m,n}.
\end{aligned}$$

This delta term enforces $n = 2i+m$. Therefore:

$$\boxed{\frac{\partial f_W(x)[0,i]}{\partial x[c_0,n]} = \begin{cases} W[0,c_0,n-2i], & n-2i \in \{0,1,2\}, \\ 0, & \text{otherwise}, \end{cases} \qquad i \in \{0,1,2\},\ c_0 \in \{1,\dots,5\},\ n \in \{1,\dots,7\}.}$$

Hence,

$$\boxed{\frac{\partial \boldsymbol{f_W(x)}}{\partial \boldsymbol{x}} \in \mathbb{R}^{1\times3\times5\times7}.}$$

—

(iv) Gradient of the Loss $\frac{\partial \ell}{\partial \boldsymbol{W}}$ Suppose the loss ℓ depends on the convolutional output $f_W(x)$, and we are given

$$\frac{\partial \ell}{\partial \boldsymbol{f_W(x)}} \in \mathbb{R}^{1\times3}.$$

By the chain rule,

$$\begin{aligned}
\frac{\partial \ell}{\partial \boldsymbol{W[0,c,m]}} &= \sum_{i=0}^{2} \frac{\partial \ell}{\partial \boldsymbol{f_W(x)[0,i]}} \frac{\partial \boldsymbol{f_W(x)[0,i]}}{\partial \boldsymbol{W[0,c,m]}} \\
&= \sum_{i=0}^{2} \frac{\partial \ell}{\partial \boldsymbol{f_W(x)[0,i]}}\, x[c,\ 2i+m].
\end{aligned}$$

Thus,

$$\boxed{\left(\frac{\partial \ell}{\partial W}\right)[0,0,c,m] = \sum_{i=0}^{2} \frac{\partial \ell}{\partial \boldsymbol{f_W(x)[0,i]}}\, x[c,\ 2i+m], \quad c \in \{1,\dots,5\},\ m \in \{0,1,2\}.}$$

The resulting tensor has shape

$$\boxed{\frac{\partial \ell}{\partial W} \in \mathbb{R}^{1\times1\times5\times3}.}$$

—

Comparison with the Forward Expression From part (i):

$$f_W(x)[0,i] = \sum_{c=1}^{5} \sum_{m=0}^{2} W[0,c,m]\, x[c, 2i+m].$$

From part (iv):

$$\frac{\partial \ell}{\partial W[0,c,m]} = \sum_{i=0}^{2} \frac{\partial \ell}{\partial f_W(x)[0,i]}\, x[c, 2i+m].$$

Similarities:

- Both expressions have identical summation structure over i and m.
- Both are correlations between two sequences — in the forward pass, between input and weights; in the backward pass, between input and loss gradients.

Differences:

- In the forward pass, W is fixed and x is variable; in the backward pass, x is fixed and $\frac{\partial \ell}{\partial f_W(x)}$ replaces the activations.
- The backward expression performs a *cross-correlation* between the input and the error signal, producing gradients of the same shape as W.

—

1.2.1 Recurrent Neural Network (30 pts, Part 1)

We are given the following recurrence:

$$c[t] = \sigma(W_c x[t] + W_h h[t-1]) \tag{1}$$

$$h[t] = c[t] \odot h[t-1] + (1 - c[t]) \odot W_x x[t] \tag{2}$$

where σ is the elementwise sigmoid, $x[t] \in \mathbb{R}^n$, $h[t] \in \mathbb{R}^m$, $W_c \in \mathbb{R}^{m\times n}$, $W_h \in \mathbb{R}^{m\times m}$, $W_x \in \mathbb{R}^{m\times n}$, and $\odot$ is elementwise multiplication. The initial hidden state is $h[0] = 0$.

—

(a) (4 pts) Diagram

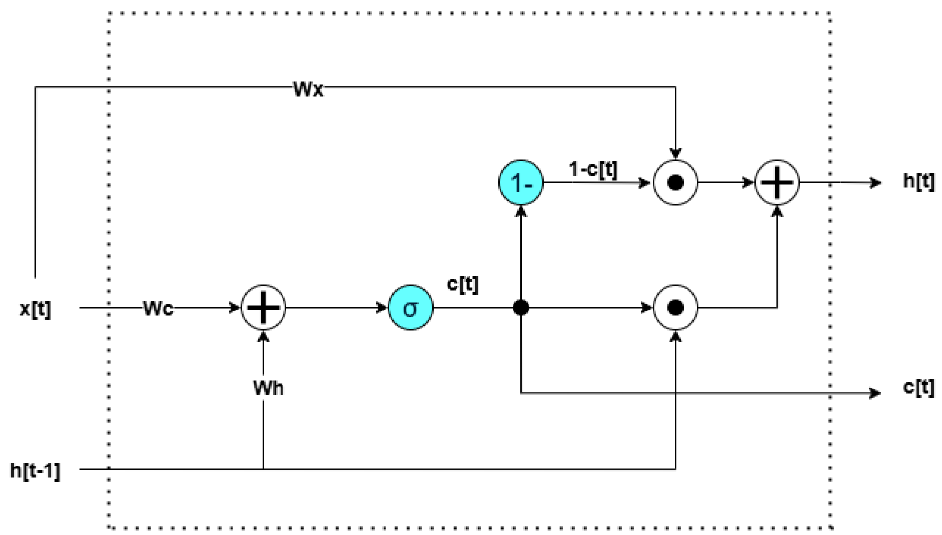


Figure 1:

—

(b) Dimension of $c[t]$ (1 pt)

From Eq. (1):

$$c[t] = \sigma(W_c x[t] + W_h h[t-1]).$$

Let us check dimensions term by term:

- $x[t] \in \mathbb{R}^n$ (the input at time t has dimension n).
- $W_c \in \mathbb{R}^{m \times n}$ (maps input into the hidden dimension).
- Therefore:
$$W_c x[t] \in \mathbb{R}^{m \times n} \cdot \mathbb{R}^n = \mathbb{R}^m.$$
- $h[t-1] \in \mathbb{R}^m$ (the hidden state at the previous step).
- $W_h \in \mathbb{R}^{m \times m}$ (maps hidden state back into hidden dimension).
- Therefore:
$$W_h h[t-1] \in \mathbb{R}^{m \times m} \cdot \mathbb{R}^m = \mathbb{R}^m.$$
- Adding the two terms:
$$W_c x[t] + W_h h[t-1] \in \mathbb{R}^m.$$
- Applying the elementwise sigmoid σ preserves dimension.

Thus,

$$\boxed{c[t] \in \mathbb{R}^m.}$$

—

(c) Gradient w.r.t. W_x (5 pts)

From the recurrence (Eq. 2):

$$h[t] = c[t] \odot h[t-1] + (1 - c[t]) \odot W_x x[t].$$

We want $\frac{\partial \ell}{\partial \boldsymbol{W_x}}$, assuming we know $\frac{\partial \ell}{\partial \boldsymbol{h[t]}}$ for all $t = 1, \ldots, K$.

- Isolate the W_x term. Only the second part of (2) depends on W_x:
$$h[t] \supset (1 - c[t]) \odot (W_x x[t]).$$
Here $(1 - c[t]) \in \mathbb{R}^m$, $W_x \in \mathbb{R}^{m \times n}$, $x[t] \in \mathbb{R}^n$, hence $W_x x[t] \in \mathbb{R}^m$.
- Write elementwise expression. For each hidden dimension $i = 1, \ldots, m$:
$$h_i[t] = (1 - c_i[t]) \cdot \left(\sum_{j=1}^{n} W_{x,ij} \, x_j[t] \right).$$
- Differentiate w.r.t. $W_{x,ij}$.
$$\frac{\partial \boldsymbol{h_i[t]}}{\partial \boldsymbol{W_{x,ij}}} = (1 - c_i[t]) \cdot x_j[t].$$
- Apply chain rule with loss. The gradient of the loss w.r.t. $W_{x,ij}$ is:
$$\frac{\partial \ell}{\partial \boldsymbol{W_{x,ij}}} = \sum_{t=1}^{K} \sum_{i=1}^{m} \frac{\partial \ell}{\partial \boldsymbol{h_i[t]}} \cdot \frac{\partial \boldsymbol{h_i[t]}}{\partial \boldsymbol{W_{x,ij}}}.$$
Substituting:
$$\frac{\partial \ell}{\partial \boldsymbol{W_{x,ij}}} = \sum_{t=1}^{K} \frac{\partial \ell}{\partial \boldsymbol{h_i[t]}} \cdot (1 - c_i[t]) \, x_j[t].$$

- Collect into matrix form. If $\frac{\partial \ell}{\partial \boldsymbol{h[t]}} \in \mathbb{R}^m$ is a column vector, then the contribution at time t is

$$\left(\frac{\partial \ell}{\partial \boldsymbol{h[t]}} \odot (1 - c[t]) \right) x[t]^\top \quad \in \mathbb{R}^{m \times n}.$$

- Final result. Summing across all time steps:

$$\boxed{\frac{\partial \ell}{\partial \boldsymbol{W_x}} = \sum_{t=1}^{K} \left(\frac{\partial \ell}{\partial \boldsymbol{h[t]}} \odot (1 - c[t]) \right) x[t]^\top}$$

which has shape $\mathbb{R}^{m \times n}$, the same as W_x.

(d) (2 pts) Vanishing/exploding gradients

Yes, this network can suffer from vanishing or exploding gradients, because:

- The recurrence depends multiplicatively on $h[t-1]$, so repeated multiplication through time can shrink gradients to zero (vanishing) or grow them without bound (exploding).
- The gating $c[t]$ may alleviate this somewhat (since it controls how much of $h[t-1]$ is retained), but it does not completely eliminate the problem like more sophisticated gating mechanisms (e.g. LSTMs or GRUs).

1.2.2 AttentionRNN (20 pts)

We define the AttentionRNN(2) as follows:

$$q_0[t], q_1[t], q_2[t] = Q_0 x[t],\ Q_1 h[t-1],\ Q_2 h[t-2] \tag{3}$$

$$k_0[t], k_1[t], k_2[t] = K_0 x[t],\ K_1 h[t-1],\ K_2 h[t-2] \tag{4}$$

$$v_0[t], v_1[t], v_2[t] = V_0 x[t],\ V_1 h[t-1],\ V_2 h[t-2] \tag{5}$$

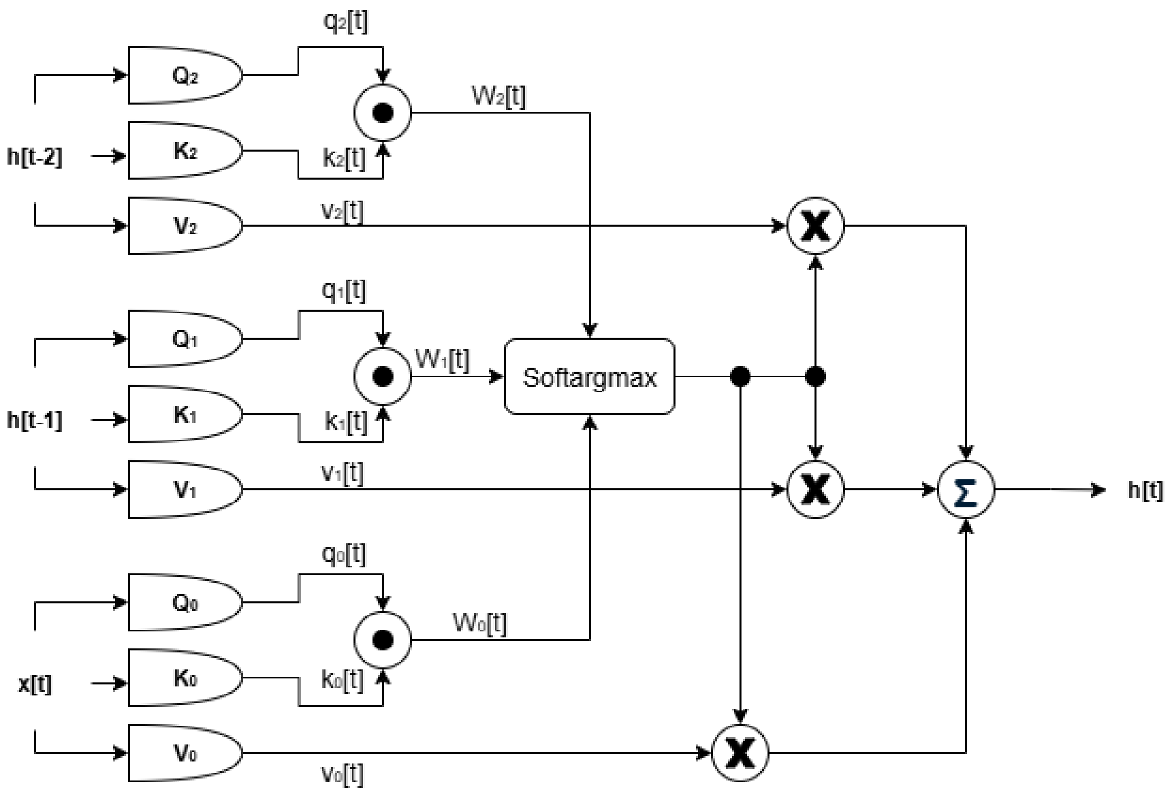
$$w_i[t] = q_i[t]^\top k_i[t] \tag{6}$$

$$a[t] = \text{softargmax}\big([w_0[t], w_1[t], w_2[t]]\big) \tag{7}$$

$$h[t] = \sum_{i=0}^{2} a_i[t]\, v_i[t]. \tag{8}$$

Here $x[t], h[t] \in \mathbb{R}^n$ and $Q_i, K_i, V_i \in \mathbb{R}^{n \times n}$. We define $h[t] = 0$ for $t < 1$ (base case, can be ignored for derivations).

(a) (4 pts) Diagram



(b) Dimension of a$[t]$

—

Step 1: Compute scalar compatibility scores. For each $i \in \{0, 1, 2\}$, we compute:

$$w_i[t] = q_i[t]^\top k_i[t],$$

which is an inner product between two n-dimensional vectors, so

$$\boxed{w_i[t] \in \mathbb{R}}.$$

Stacking all three scores:

$$w[t] = \begin{bmatrix} w_0[t] \\ w_1[t] \\ w_2[t] \end{bmatrix} \in \mathbb{R}^{3\times 1}.$$

—

Step 2: Apply the softmax to obtain attention weights. The attention vector $a[t]$ is computed as

$$a[t] = \text{softargmax}(w[t]) = \text{softmax}(w[t]) = \begin{bmatrix} a_0[t] \\ a_1[t] \\ a_2[t] \end{bmatrix},$$

where each entry

$$a_i[t] = \frac{e^{w_i[t]}}{\sum_{j=0}^{2} e^{w_j[t]}}, \qquad a_i[t] \geq 0, \qquad \sum_{i=0}^{2} a_i[t] = 1.$$

—

Step 3: Determine the dimension. Since there are three scalar scores (w_0, w_1, w_2) corresponding respectively to

$$x[t], \; h[t-1], \text{ and } h[t-2],$$

the attention weight vector has one element per candidate:

$$\boxed{a[t] \in \mathbb{R}^{3\times 1}.}$$

—

Step 4: Interpretation. The attention vector $a[t]$ provides normalized coefficients expressing the relative importance of the current input and the two most recent hidden states when forming the next hidden state:

$$h[t] = \sum_{i=0}^{2} a_i[t]\, v_i[t].$$

Hence $a_0[t], a_1[t], a_2[t]$ act as weighting factors for $x[t]$, $h[t-1]$, and $h[t-2]$, respectively.
—

Final Answer.

$$\boxed{a[t] = \begin{bmatrix} a_0[t] \\ a_1[t] \\ a_2[t] \end{bmatrix} \in \mathbb{R}^{3\times 1}, \qquad \sum_{i=0}^{2} a_i[t] = 1.}$$

(c) (3 pts) AttentionRNN(k)

Generalizing to the last k hidden states:

$$q_i[t] = Q_i h[t-i], \quad k_i[t] = K_i h[t-i], \quad v_i[t] = V_i h[t-i], \quad i = 0, 1, \ldots, k,$$

where $h[t]$ is defined to be $x[t]$ when $i = 0$.

Scores:

$$w_i[t] = q_i[t]^\top k_i[t], \quad i = 0, 1, \ldots, k.$$

Attention:

$$a[t] = \text{softargmax}([w_0[t], w_1[t], \ldots, w_k[t]]) \in \mathbb{R}^{k+1}.$$

Hidden state:

$$h[t] = \sum_{i=0}^{k} a_i[t]\, v_i[t].$$

—

(d) (3 pts) AttentionRNN(∞)

We can extend this to all past hidden states by parameter sharing:

$$q_i[t] = Qh[t-i], \quad k_i[t] = Kh[t-i], \quad v_i[t] = Vh[t-i], \quad i \geq 0.$$

Scores:

$$w_i[t] = q_i[t]^\top k_i[t], \quad i = 0, 1, 2, \ldots$$

Attention:

$$a[t] = \text{softargmax}([w_0[t], w_1[t], w_2[t], \ldots]).$$

Hidden state:

$$h[t] = \sum_{i=0}^{t} a_i[t]\, v_i[t].$$

This allows the network to attend over all past states with tied parameters (Q, K, V).
—

(e) (5 pts) Derivative $\frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[t-1]}}$ for AttentionRNN(2)

We recall the recurrence:

$$h[t] = a_0[t]v_0[t] + a_1[t]v_1[t] + a_2[t]v_2[t],$$

where

$$v_0[t] = V_0x[t], \quad v_1[t] = V_1h[t-1], \quad v_2[t] = V_2h[t-2].$$

The attention weights are:

$$a[t] = \text{softargmax}\left([w_0[t], w_1[t], w_2[t]]\right),$$

with scores

$$w_i[t] = q_i[t]^\top k_i[t].$$

In particular,

$$q_1[t] = Q_1h[t-1], \qquad k_1[t] = K_1h[t-1], \qquad v_1[t] = V_1h[t-1].$$

—

- **Step 1: Identify dependencies.** The current hidden state $h[t]$ depends on $h[t-1]$ in two ways:
  1. *Directly through $v_1[t] = V_1h[t-1]$.*
  2. *Indirectly through $a[t]$, since $a[t]$ depends on the scores $w_1[t]$ which in turn depend on $h[t-1]$.*
- **Step 2: Differentiate $h[t]$ w.r.t. $h[t-1]$.** Differentiating term by term:

$$\frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[t-1]}} = \underbrace{\frac{\partial(\boldsymbol{a_1[t]v_1[t]})}{\partial \boldsymbol{h[t-1]}}}_{\text{direct + indirect}} + \underbrace{\frac{\partial(\boldsymbol{a_0[t]v_0[t]})}{\partial \boldsymbol{h[t-1]}}}_{\text{indirect only}} + \underbrace{\frac{\partial(\boldsymbol{a_2[t]v_2[t]})}{\partial \boldsymbol{h[t-1]}}}_{\text{indirect only}}.$$

- **Step 3: Expand the direct contribution.** For the middle term:

$$\frac{\partial(\boldsymbol{a_1[t]v_1[t]})}{\partial \boldsymbol{h[t-1]}} = a_1[t] \cdot \frac{\partial \boldsymbol{v_1[t]}}{\partial \boldsymbol{h[t-1]}} + v_1[t] \cdot \frac{\partial \boldsymbol{a_1[t]}}{\partial \boldsymbol{h[t-1]}}.$$

  Since $v_1[t] = V_1h[t-1]$,

$$\frac{\partial \boldsymbol{v_1[t]}}{\partial \boldsymbol{h[t-1]}} = V_1.$$

  Hence:

$$\frac{\partial(\boldsymbol{a_1[t]v_1[t]})}{\partial \boldsymbol{h[t-1]}} = a_1[t]V_1 + v_1[t] \cdot \frac{\partial \boldsymbol{a_1[t]}}{\partial \boldsymbol{h[t-1]}}.$$

- **Step 4: Expand the indirect contributions.** The other two terms ($a_0[t]v_0[t]$ and $a_2[t]v_2[t]$) do not depend on $h[t-1]$ directly, but only through their attention weights:

$$\frac{\partial(\boldsymbol{a_0[t]v_0[t]})}{\partial \boldsymbol{h[t-1]}} = v_0[t] \cdot \frac{\partial \boldsymbol{a_0[t]}}{\partial \boldsymbol{h[t-1]}}, \qquad \frac{\partial(\boldsymbol{a_2[t]v_2[t]})}{\partial \boldsymbol{h[t-1]}} = v_2[t] \cdot \frac{\partial \boldsymbol{a_2[t]}}{\partial \boldsymbol{h[t-1]}}.$$

- **Step 5: Differentiate the attention weights.** The attention weights are obtained from the softmax:

$$a_i[t] = \frac{\exp(w_i[t])}{\sum_{j=0}^{2}\exp(w_j[t])}.$$

  To compute its derivative, we apply the quotient rule. For $i = j$:

$$\frac{\partial \boldsymbol{a_i[t]}}{\partial \boldsymbol{w_i[t]}} = \frac{\exp(w_i[t]) \cdot \sum_{j=0}^{2}\exp(w_j[t]) - \exp(w_i[t]) \cdot \exp(w_i[t])}{\left(\sum_{j=0}^{2}\exp(w_j[t])\right)^2}.$$

  Simplifying:

$$\frac{\partial \boldsymbol{a_i[t]}}{\partial \boldsymbol{w_i[t]}} = a_i[t]\big(1 - a_i[t]\big).$$

For $i \neq j$:

$$\frac{\partial \boldsymbol{a_i[t]}}{\partial \boldsymbol{w_j[t]}} = \frac{0 \cdot \sum_{u=0}^{2} \exp(w_u[t]) - \exp(w_i[t]) \cdot \exp(w_j[t])}{\left(\sum_{u=0}^{2} \exp(w_u[t])\right)^2}.$$

Simplifying:

$$\frac{\partial \boldsymbol{a_i[t]}}{\partial \boldsymbol{w_j[t]}} = -a_i[t]a_j[t].$$

Both cases can be summarized compactly using the *Kronecker delta*:

$$\frac{\partial \boldsymbol{a_i[t]}}{\partial \boldsymbol{w_j[t]}} = a_i[t](\delta_{ij} - a_j[t]),$$

where

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

- **Step 6: Apply chain rule.** Since only $w_1[t]$ depends on $h[t-1]$, we have

$$\frac{\partial \boldsymbol{a_i[t]}}{\partial \boldsymbol{h[t-1]}} = \frac{\partial \boldsymbol{a_i[t]}}{\partial \boldsymbol{w_1[t]}} \cdot \frac{\partial \boldsymbol{w_1[t]}}{\partial \boldsymbol{h[t-1]}}.$$

- **Step 7: Differentiate the score $w_1[t]$.** By definition:

$$w_1[t] = (Q_1 h[t-1])^\top (K_1 h[t-1]) = h[t-1]^\top Q_1^\top K_1 h[t-1].$$

Differentiating w.r.t. $h[t-1]$:

$$\frac{\partial \boldsymbol{w_1[t]}}{\partial \boldsymbol{h[t-1]}} = Q_1^\top k_1[t] + K_1^\top q_1[t],$$

where $q_1[t] = Q_1 h[t-1]$ and $k_1[t] = K_1 h[t-1]$.

- **Step 8: Combine results.** Substituting back into the derivative of $h[t]$:

$$\frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[t-1]}} = a_1[t]V_1 + v_1[t] \cdot \frac{\partial \boldsymbol{a_1[t]}}{\partial \boldsymbol{w_1[t]}} \cdot \frac{\partial \boldsymbol{w_1[t]}}{\partial \boldsymbol{h[t-1]}} + v_0[t] \cdot \frac{\partial \boldsymbol{a_0[t]}}{\partial \boldsymbol{w_1[t]}} \cdot \frac{\partial \boldsymbol{w_1[t]}}{\partial \boldsymbol{h[t-1]}} + v_2[t] \cdot \frac{\partial \boldsymbol{a_2[t]}}{\partial \boldsymbol{w_1[t]}} \cdot \frac{\partial \boldsymbol{w_1[t]}}{\partial \boldsymbol{h[t-1]}}.$$

Using the compact softmax derivative:

$$\frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[t-1]}} = a_1[t]V_1 + \sum_{i=0}^{2} v_i[t]\, a_i[t](\delta_{i1} - a_1[t]) \cdot \left(Q_1^\top k_1[t] + K_1^\top q_1[t]\right)^\top.$$

—

Final Expression:

$$\boxed{\frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[t-1]}} = a_1[t]V_1 + \sum_{i=0}^{2} v_i[t]\, a_i[t](\delta_{i1} - a_1[t]) \cdot \left(Q_1^\top k_1[t] + K_1^\top q_1[t]\right)^\top}$$

This captures both the *direct contribution* via $v_1[t] = V_1 h[t-1]$ and the *indirect contribution* via the softmax attention weights $a[t]$.

—

(f) (2 pts) Backpropagation for $\frac{\partial \ell}{\partial \boldsymbol{h[T]}}$ in AttentionRNN(k)

We want the gradient of the loss ℓ with respect to some hidden state $h[T]$. We are given: - $\frac{\partial \ell}{\partial \boldsymbol{h[t]}}$ for all $t > T$ (these are the "direct" gradients from the loss into later hidden states), - and $\frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[T]}}$ for all $t > T$ (these describe how $h[T]$ influences later states through recurrence).

—

- **Step 1: Direct effect at time T.** If $h[T]$ itself contributes directly to the loss (e.g. via an output layer at time T), then that contributes:

$$\frac{\partial \ell}{\partial \boldsymbol{h[T]}} \supset \left(\frac{\partial \ell}{\partial \boldsymbol{h[T]}}\right)_{\text{direct}}.$$

In many setups, the direct gradient may be zero unless $h[T]$ is explicitly used to compute ℓ.

- **Step 2: Indirect effect via future steps.** Every later hidden state $h[t]$ with $t > T$ depends on $h[T]$ (possibly through multiple paths). By the chain rule:

$$\left(\frac{\partial \ell}{\partial \boldsymbol{h[T]}}\right)_{\text{indirect}} = \sum_{t=T+1}^{K} \frac{\partial \ell}{\partial \boldsymbol{h[t]}} \cdot \frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[T]}}.$$

- **Step 3: Expand $\frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[T]}}$.** The term $\frac{\partial h[t]}{\partial h[T]}$ itself is recursive:

$$\frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[T]}} = \frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[t-1]}} \cdot \frac{\partial \boldsymbol{h[t-1]}}{\partial \boldsymbol{h[T]}} + \frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[t-2]}} \cdot \frac{\partial \boldsymbol{h[t-2]}}{\partial \boldsymbol{h[T]}} + \cdots + \frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[T]}} \cdot I.$$

For AttentionRNN(k), each $h[t]$ depends on the k previous states. So in general:

$$\frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[T]}} = \sum_{j=1}^{k} \frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[t-j]}} \cdot \frac{\partial \boldsymbol{h[t-j]}}{\partial \boldsymbol{h[T]}}.$$

- **Step 4: Combine results.** Thus, the full gradient is:

$$\frac{\partial \ell}{\partial \boldsymbol{h[T]}} = \left(\frac{\partial \ell}{\partial \boldsymbol{h[T]}}\right)_{\text{direct}} + \sum_{t=T+1}^{K} \frac{\partial \ell}{\partial \boldsymbol{h[t]}} \cdot \frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[T]}}.$$

—

Compact expression:

$$\boxed{\frac{\partial \ell}{\partial \boldsymbol{h[T]}} = \frac{\partial \ell}{\partial \boldsymbol{h[T]}} \text{ (direct term)} + \sum_{t=T+1}^{K} \frac{\partial \ell}{\partial \boldsymbol{h[t]}} \cdot \frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[T]}}}$$

—

Expanded view for AttentionRNN(k): Since each $h[t]$ depends on the k previous states,

$$\frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[T]}} = \begin{cases} \sum_{j=1}^{k} \frac{\partial \boldsymbol{h[t]}}{\partial \boldsymbol{h[t-j]}} \cdot \frac{\partial \boldsymbol{h[t-j]}}{\partial \boldsymbol{h[T]}}, & \text{if } T \le t-1, \\ 0, & \text{otherwise.} \end{cases}$$

So the loss gradient w.r.t. $h[T]$ propagates backward through all possible paths of length up to k, exactly like standard backpropagation-through-time but with k-step connections instead of only one.

—

1.3 Debugging loss curves

1. What caused the spikes on the left? (1 pt)

The spikes in the early epochs are caused by *exploding gradients* in the RNN during training. RNNs repeatedly multiply by weight matrices through time steps, which can cause the gradient values to grow very large. When this happens, the parameter update becomes extremely large, temporarily making the loss shoot up. This is common in sequence models without gradient clipping.

2. How can they be higher than the initial value of the loss? (1 pt)

The initial loss corresponds to predictions from a randomly initialized network, which are essentially "uninformed guesses." Once training starts, a very large weight update (caused by exploding gradients) can make the network's predictions much worse than random, producing loss values higher than the starting baseline. In other words, poor updates can push the model into a bad region of parameter space before it recovers.

3. What are some ways to fix them? (1 pt)

Several standard techniques can reduce or eliminate these spikes:

- **Gradient clipping** (most common): cap the maximum norm of gradients to prevent runaway updates.
- **Smaller learning rate**: reduces the size of parameter updates.
- **Weight initialization strategies** (e.g., orthogonal initialization for recurrent weights) to reduce instability.
- **Use gated RNNs (LSTMs/GRUs)** instead of vanilla RNNs, since gates control gradient flow better.

4. Why are the loss and accuracy at these values before training starts? (2 pts)

Before training begins (epoch 0), the model weights are randomly initialized. Predictions are therefore random guesses across the possible output classes.

If this is a classification task (like sequence classification in the notebook), then:

- The initial accuracy will be roughly uniform guessing, i.e.

$$\text{accuracy} \approx \frac{1}{\text{num classes}}.$$

- The initial loss will be close to the cross-entropy of the uniform distribution, i.e.

$$\ell \approx \log(\text{num classes}).$$

In the given graph it seems there are 4 classes:

$$\ell \approx \log 4 \approx 1.386, \qquad \text{accuracy} \approx \tfrac{1}{4} = 25\%.$$