Homework 2: Convolutional Neural Networks and Recurrent Neural Networks

CSCI-GA 2572 Deep Learning

Fall 2024

1 Theory (50pt)

1.1 Convolutional Neural Netoworks (15 pts)

(a) (1 pts)

Given stride s and convolution size of $k_h \times k_w$ on input of $n_h \times n_w$ with zero padding will be: $(([n_h - k_h]/s) + 1) \times (([n_w - k_w]/s) + 1)$.

Final output image will have size of $(([17-3]/2)+1) \times (([11-5]/2)+1) = 8 \times 4$.

(b) (2 pts)

- H and W are the input height and width.
- HK and WK are the kernel height and width.
- P is the padding size.
- S is the stride.
- D is the dilation (which controls the spacing between kernel elements).
- F is the number of filters.

$$\text{Output Dimension} = F \times \left(\frac{H+2P-D\times(HK-1)-1}{S}+1\right) \times \left(\frac{W+2P-D\times(WK-1)-1}{S}+1\right)$$

(c) (12 pts)

(i) For 1 dimensional convolutions,
Output length $= \left\lfloor \frac{\text{Input\_length}+2\times\text{padding}-\text{dilation}\times(\text{kernel\_size}-1)-1}{\text{stride}} \right\rfloor + 1$
Input length = 11, Kernel size $K = 3$, Stride $S = 2$, Dilation $D = 1$ and Padding $P = 0$

$$\textbf{Output length} = \left\lfloor \frac{11+0-1\times(3-1)-1}{2} \right\rfloor + 1 = \left\lfloor \frac{8}{2} \right\rfloor + 1 = 4 + 1 = 5$$

Therefore, the output $f_W(x)$ has a dimension of $\mathbb{R}^5$.

For the expression of the output elements, we define:

$$x[c,p] \text{ with } c = 1,\ldots,7 \text{(channels) and } p = 1,\ldots,11 \text{(positions)}$$

$$W[i,c,m] \text{ with } i = 1,\ c = 1,\ldots,7 \text{ and } m = 1,2,3 \text{(kernel positions)}$$

The position in x corresponding to output index k and kernel index m is:

$$p = (k-1) \times \text{stride} + m$$

Thus, the expression for $f_W(x)[k]$ is:

$$f_W(x)[k] = \sum_{i=1}^{1} \sum_{c=1}^{7} \sum_{m=1}^{3} x[c,(k-1)\times 2 + m] \times W[i,c,m]$$

The output is $f_W(x) \in \mathbb{R}^5$, with elements

$$\boxed{f_W(x)[k] = \sum_{i=1}^{1} \sum_{c=1}^{7} \sum_{m=1}^{3} x[c,(k-1)\times 2 + m] \times W[i,c,m]}$$

(ii) (4 pts)

Dimension of $f_W(x) \in \mathbb{R}^5$ and dimension of $W \in \mathbb{R}^{1\times 7\times 3}$. Which means $\frac{\partial f_W(x)}{\partial W}$ will $\in \mathbb{R}^{5\times 1\times 7\times 3}$.

And we can represent $f_W(x) \in \mathbb{R}^5$, with elements

$$f_W(x)[k] = \sum_{i=1}^{1} \sum_{c=1}^{7} \sum_{m=1}^{3} x[c,(k-1)\times 2 + m] \times W[i,c,m]$$

s Since s$f_W(x)[k]$ is linear with respect to $W[i,c,m]$, partial derivative of $f_W(x)[k]$ with respect to $W[i,c,m]$ is

$$\boxed{\frac{\partial f_W(x)[k]}{\partial W[i,c,m]} = x[c,(k-1)\times 2 + m] \text{ with } \frac{\partial f_W(x)[k]}{\partial W[i,c,m]} \in \mathbb{R}^{5\times 1\times 7\times 3}}$$

Thus, an expression for the values of the derivative is $\frac{\partial f_W(x)[k]}{\partial W[i,c,m]} = x[c,(k-1)\times 2 + m]$ where k ranges from [1,..,5], c ranges from [1,..,7], $i = 1$ and m ranges from [1,..,3] which makes $\frac{\partial f_W(x)}{\partial W} \in \mathbb{R}^{5\times 1\times 7\times 3}$.

(iii) (4 pts)

$$f_W(x)[k] = \sum_{i=1}^{1} \sum_{c=1}^{7} \sum_{m=1}^{3} x[c,(k-1)\times 2 + m] \times W[i,c,m]$$

Each output element $f_W(x)[k]$ depends only on a subset of the input x, specifically the positions corresponding to the convolution window at step k. $f_W(x) \in \mathbb{R}^5$ and $x \in \mathbb{R}^{7\times 11}$ which makes $\frac{\partial f_W(x)}{\partial x} \in \mathbb{R}^{5\times 7\times 11}$.

For each output index k, input channel c, and input position p:

The derivative $\frac{\partial f_W(x)[k]}{\partial x[c,p]}$ is non-zero only when $p = (k-1) \times 2 + m$ for $m = 1, 2, 3$.

Again as $f_W(x)[k]$ is linear with respect to $x[c,p]$ the $\frac{\partial f_W(x)[k]}{\partial x[c,p]}$ is simply, $\frac{\partial f_W(x)[k]}{\partial x[c,p]} = W[i,c,m]$. We can substitute m as $p-(k-1)\times 2$.

For $i = 1, k = 1$ to $5, c = 1$ to $7, p = 1$ to 11 :

$$\frac{\partial f_W(x)[k]}{\partial x[c,p]} = \begin{cases} W[i,c,p-(k-1)\times 2], & \text{if } p-(k-1)\times 2 \in \{1,2,3\} \\ 0, & \text{otherwise} \end{cases}$$

Dimension of partial derivative is

$$\frac{\partial f_W(x)}{\partial x} \in \mathbb{R}^{5\times 7\times 11}$$

(iv) (5 pts)

The weight W has dimensions $\mathbb{R}^{1\times 7\times 3}$. l is a scalar value which makes the gradient $\frac{\partial \ell}{\partial W} \in \mathbb{R}^{1\times 7\times 3}$

Therefore:

$$\frac{\partial \ell}{\partial W} \in \mathbb{R}^{1\times 7\times 3}$$

Expression for $\frac{\partial \ell}{\partial W}$:

- The gradient of the loss with respect to the output of the convolutional layer: $\frac{\partial \ell}{\partial f_W(x)} \in \mathbb{R}^5$.
- The previously derived expression for $\frac{\partial f_W(x)}{\partial W}$:

$$\frac{\partial f_W(x)[k]}{\partial W[i,c,m]} = x[c,(k-1)\times 2+m]$$

Using the chain rule for gradients:

$$\frac{\partial \ell}{\partial W[i,c,m]} = \sum_{k=1}^{5} \frac{\partial \ell}{\partial f_W(x)[k]} \times \frac{\partial f_W(x)[k]}{\partial W[i,c,m]}$$

$$\frac{\partial \ell}{\partial W[i,c,m]} = \sum_{k=1}^{5} \frac{\partial \ell}{\partial f_W(x)[k]} \times x[c,(k-1)\times 2+m]$$

- The gradient $\frac{\partial \ell}{\partial W} \in \mathbb{R}^{1\times 7\times 3}$,
- With elements:

$$\frac{\partial \ell}{\partial W[i,c,m]} = \sum_{k=1}^{5} \frac{\partial \ell}{\partial f_W(x)[k]} \times x[c,(k-1)\times 2+m]$$

- **Similarities** : Both expressions involve the input x evaluated at positions that depend on the output index k, the stride, and the kernel index m.
- **Differences** : In $f_W(x)$, the summation is over the input channels c and kernel positions m for each output position k. In $\frac{\partial \ell}{\partial W}$, the summation is over the output positions k for each weight position (i, c, m).
- **Forward pass**: At each output position k, the convolution operation aggregates information from the input using the weights. This process slides the kernel across the input, combining input values and weights to produce the output.
- **Gradient Computation** : The gradient with respect to the weights accumulates the influence of each output gradient $\frac{\partial \ell}{\partial f_W(x)[k]}$ on the weights. It effectively reverses the convolution process, using the gradients from the output to adjust the weights based on how much each weight contributed to the loss.

$$Gate(h[t-1], x[t]) = c[t] = \sigma(W_c x[t] + W_h h[t-1])$$
$$Pred(h[t-1], x[t], c[t]) = h[t] = c[t] \odot h[t-1] + (1 - c[t]) \odot W_x x[t]$$

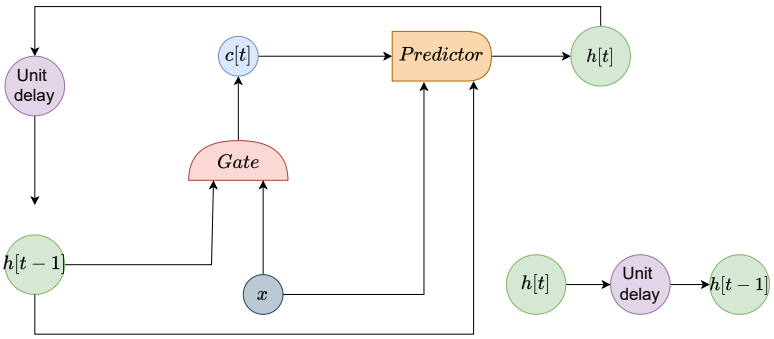

Figure 1: RNN diagram of 1.2.1 a

1.2 Recurrent Neural Networks (30 pts)

1.2.1 Part 1

(a) (4 pts)

(b) (1pts) What is the dimension of $c[t]$?

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

$x[t] \in \mathbb{R}^n$, $h[t-1] \in \mathbb{R}^m$, $W_c \in \mathbb{R}^{m \times n}$ and $W_h \in \mathbb{R}^{m \times m}$.

1. $W_c x[t]$:
   - W_c is $m \times n$
   - $x[t]$ is $n \times 1$
   - The product $W_c x[t]$ results in a vector of dimension $m \times 1$
2. $W_h h[t-1]$:
   - W_h is $m \times m$
   - $h[t-1]$ is $m \times 1$
   - The product $W_h h[t-1]$ results in a vector of dimension $m \times 1$
3. Summation of $W_c x[t]$ and $W_h h[t-1]$:
   - $W_c x[t] + W_h h[t-1]$ is an element-wise addition of two $m \times 1$ vectors, resulting in an $m \times 1$ vector.

4. Sigmoid function σ:

   - The sigmoid function is applied element-wise to the $m \times 1$ vector.
   - The result $c[t]$ remains an $m \times 1$ vector.

$$\text{The vector } c[t] \text{ has dimension } c[t] \in \mathbb{R}^m.$$

(c) (3 pts) Derive expressions for $\frac{\partial h[t]}{\partial h[t-1]}$ and $\frac{\partial c[t]}{\partial h[t-1]}$

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

where: $x[t] \in \mathbb{R}^n$, $h[t-1] \in \mathbb{R}^m$, $W_c \in \mathbb{R}^{m \times n}$ and $W_h \in \mathbb{R}^{m \times m}$.

1) $\frac{\partial c[t]}{\partial h[t-1]}$:

$$c[t] = \sigma(s[t]), \quad \text{where } s[t] = W_c x[t] + W_h h[t-1]$$

- For $\frac{\partial c[t]}{\partial s[t]}$, the derivative of the sigmoid function $\sigma(s)$ is:

$$\sigma'(s) = \sigma(s)(1 - \sigma(s))$$

  Since $c[t] = \sigma(s[t])$ and sigmoid is element wise function the Jacobian will only have non zero entries at diagonal. with $\frac{\partial c[t]}{\partial s[t]} \in \mathbb{R}^{m \times m}$:

$$\frac{\partial c[t]}{\partial s[t]} = \text{diag}(c[t] \odot (1 - c[t]))$$

- For $\frac{\partial s[t]}{\partial h[t-1]}$, both $s[t]$ and $h[t-1]$ are $\in \mathbb{R}^m$ which makes $\frac{\partial s[t]}{\partial h[t-1]} \in \mathbb{R}^{m \times m}$ Since $s[t] = W_c x[t] + W_h h[t-1]$:

$$\frac{\partial s[t]}{\partial h[t-1]} = W_h$$

- Finally, applying chain rule and both $h[t-1]$ and $c[t]$ are $\in \mathbb{R}^m$ which makes $\frac{\partial c[t]}{\partial h[t-1]} \in \mathbb{R}^{m \times m}$

$$\boxed{\frac{\partial c[t]}{\partial h[t-1]} = \frac{\partial c[t]}{\partial s[t]} \cdot \frac{\partial s[t]}{\partial h[t-1]} = \text{diag}(c[t] \odot (1 - c[t]))W_h}$$

2) $\frac{\partial h[t]}{\partial h[t-1]}$:

$$h[t] = c[t] \odot h[t-1] + (1 - c[t]) \odot W_x x[t] \text{ and } h[t] \in \mathbb{R}^m$$

- Differentiating the term $c[t] \odot h[t-1]$:

$$\frac{\partial}{\partial h[t-1]}(c[t] \odot h[t-1]) = \text{diag}(h[t-1])\frac{\partial c[t]}{\partial h[t-1]} + \text{diag}(c[t])$$

- Differentiating the term $(1-c[t]) \odot W_x x[t]$:

$$\frac{\partial}{\partial h[t-1]}((1-c[t]) \odot W_x x[t]) = -\text{diag}(W_x x[t])\frac{\partial c[t]}{\partial h[t-1]}$$

- Combining the partial derivatives:

$$\frac{\partial h[t]}{\partial h[t-1]} = \text{diag}(c[t]) + \text{diag}(h[t-1] - W_x x[t])\frac{\partial c[t]}{\partial h[t-1]}$$

Substituting $\frac{\partial c[t]}{\partial h[t-1]}$ from 1.

$$\boxed{\frac{\partial h[t]}{\partial h[t-1]} = \text{diag}(c[t]) + \text{diag}(h[t-1] - W_x x[t])\text{diag}(c[t] \odot (1-c[t]))W_h}$$

(d) (4 pts)

Dimension of $\frac{\partial \ell}{\partial W_x}$:

Given $W_x \in \mathbb{R}^{m \times n}$ and l is scalar, the gradient $\frac{\partial \ell}{\partial W_x}$ will have the same dimensions as W_x.

$$\frac{\partial \ell}{\partial W_x} \in \mathbb{R}^{m \times n}$$

Expression for $\frac{\partial \ell}{\partial W_x}$:

We have the derivative of the loss with respect to the hidden states: $\frac{\partial \ell}{\partial h[t]}$ for $t = 1$ to K.

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

We can write $\frac{\partial \ell}{\partial W_x}$ as $\frac{\partial \ell}{\partial h[t]}\frac{\partial h[t]}{\partial W_x}$
And as t ranges from 1 to K, this effectively becomes $\frac{\partial \ell}{\partial W_x} = \sum_{t=1}^{K}\frac{\partial \ell}{\partial h[t]}\frac{\partial h[t]}{\partial W_x}$.

Now for $\frac{\partial h[t]}{\partial W_x}$, Differentiating first term in eq. 2: $c[t] \odot h[t-1]$,

$$\frac{\partial c[t]}{\partial W_x}\text{diag}(h[t-1]) + \text{diag}(c[t])\frac{\partial h[t-1]}{\partial W_x} \tag{3}$$

Differentiating second term in eq. 2: $(1-c[t]) \odot W_x x[t]$,

$$-\frac{\partial c[t]}{\partial W_x}\text{diag}(W_x x[t]) + \text{diag}(1-c[t])x[t]^\top \tag{4}$$

Now, computing $\frac{\partial c[t]}{\partial W_x}$,

$$c[t] = \sigma(s[t]), \quad \text{where } s[t] = W_c x[t] + W_h h[t-1]$$

Thus,

$$\frac{\partial c[t]}{\partial W_x} = \frac{\partial c[t]}{\partial s[t]} \frac{\partial s[t]}{\partial W_x}$$

which makes,

$$\frac{\partial c[t]}{\partial W_x} = \text{diag}(c[t] \odot (1 - c[t])) W_h \frac{\partial h[t-1]}{\partial W_x}$$

Now combining eq. 3 and eq. 4,

$$\frac{\partial h[t]}{\partial W_x} = \text{diag}(h[t-1] - W_x x[t]) \frac{\partial c[t]}{\partial W_x} + \text{diag}(c[t]) \frac{\partial h[t-1]}{\partial W_x} + \text{diag}(1 - c[t]) x[t]^\top$$

substituting $\frac{\partial c[t]}{\partial W_x}$ from above,

$$\frac{\partial h[t]}{\partial W_x} = (\text{diag}(c[t]) + \text{diag}(h[t-1] - W_x x[t]) \text{diag}(c[t] \odot (1 - c[t])) W_h) \frac{\partial h[t-1]}{\partial W_x} + \text{diag}(1 - c[t]) x[t]^\top$$

Now considering,

$$a_t = (\text{diag}(c[t]) + \text{diag}(h[t-1] - W_x x[t]) \text{diag}(c[t] \odot (1 - c[t])) W_h)$$

$$b_t = \text{diag}(1 - c[t]) x[t]^\top$$

$$\frac{\partial h[t]}{\partial W_x} = a_t \frac{\partial h[t-1]}{\partial W_x} + b_t$$

Again considering,

$$\frac{\partial h[t]}{\partial W_x} = K_t$$

We can write,

$$K_t = a_t K_{t-1} + b_t$$

We know $K_0 = 0$, thus we can write,

$$K_t = b_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} a_j \right) b_i.$$

$$\frac{\partial h[t]}{\partial W_x} = b_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} a_j \right) b_i.$$

Finally, we know,

$$\frac{\partial \ell}{\partial W_x} = \sum_{t=1}^{K} \frac{\partial \ell}{\partial h[t]} \frac{\partial h[t]}{\partial W_x}$$

which makes

$$\frac{\partial \ell}{\partial W_x} = \sum_{t=1}^{K} \frac{\partial \ell}{\partial h[t]} \left(b_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} a_j \right) b_i \right)$$

where

$$a_t = (\text{diag}(c[t]) + \text{diag}(h[t-1] - W_x x[t])\text{diag}(c[t] \odot (1 - c[t]))W_h)$$

and

$$b_t = \text{diag}(1 - c[t])x[t]^\top$$

Similarities Between Backward Pass and Forward Pass:

1. Sequential Dependency Over Time:

   - Forward Pass:
     - The hidden state $h[t]$ depends on $h[t-1]$ and the input $x[t]$.
     - Computation proceeds from $t = 1$ to $t = K$, sequentially updating $h[t]$ using $h[t-1]$.
   - Backward Pass:
     - The gradient $\frac{\partial \ell}{\partial h[t-1]}$ depends on $\frac{\partial \ell}{\partial h[t]}$ due to the recurrent connections.
     - Computation proceeds from $t = K$ down to $t = 1$, sequentially updating $\frac{\partial \ell}{\partial h[t-1]}$ using $\frac{\partial \ell}{\partial h[t]}$.

2. Use of Gates and Element-wise Operations:

   - Forward Pass:
     - The update gate $c[t]$ modulates the influence of the previous hidden state $h[t-1]$ and the new input $x[t]$.
   - Backward Pass:
     - The gradient calculations involve the same gates $c[t]$ and element-wise operations.
     - For example, the term $(c[t])$ and $(1 - c[t])$ in the gradient with respect to W_x mirrors the gating mechanism in the forward pass.

(e) (2pts)

Gradient Propagation Through Time:

- The gradient of the loss with respect to the hidden state at time $t-1$ is:

$$\frac{\partial \ell}{\partial h[t-1]} = \frac{\partial \ell}{\partial h[t]} \frac{\partial h[t]}{\partial h[t-1]}$$

- Over multiple time steps, the gradient involves a product of the Jacobians $\frac{\partial h[t]}{\partial h[t-1]}$:

$$\frac{\partial \ell}{\partial h[t_0]} = \frac{\partial \ell}{\partial h[K]} \prod_{t=t_0+1}^{K} \frac{\partial h[t]}{\partial h[t-1]}$$

- We know individual partial derivatives are

$$\frac{\partial h[t]}{\partial h[t-1]} = \text{diag}(c[t]) + \text{diag}(h[t-1] - W_x x[t])\text{diag}(c[t] \odot (1 - c[t]))W_h$$

b. Potential for Vanishing Gradients:

- **Sigmoid Activation Derivative:** The derivative $c[t] \odot (1 - c[t])$ has a maximum value of 0.25 and decreases as $c[t]$ approaches 0 or 1, leading to small gradients.
- **Diagonal Elements Less Than 1:** The matrices $\text{diag}(c[t])$ have entries between 0 and 1. Repeated multiplication of such matrices can cause the gradients to shrink exponentially.
- The product of Jacobians may have norms that decrease exponentially, causing gradients to vanish over time steps.

c. Potential for Exploding Gradients:

- **Weight Matrix W_h:** If W_h has large singular values or if the term $\text{diag}(h[t-1] - W_x x[t])$ amplifies the gradients, the product of Jacobians can lead to exponentially increasing gradients.
- **Nonlinear Interactions:** The combination of terms involving W_h and the dependencies on $h[t-1]$ can result in gradients that grow rapidly.
- The gradients can explode, leading to instability during training.

1.2.2 Part 2

(a) (1 pt) $x[t], h[t] \in \mathbb{R}^n$ and $Q_i, K_i, V_i \in \mathbb{R}^{n \times n}$ for $i = 0, 1, 2$ $h[t] = 0$ for $t < 1$

$$\begin{array}{lll} q_0[t] = Q_0 x[t] & k_0[t] = K_0 x[t] & v_0[t] = V_0 x[t] \\ q_1[t] = Q_1 h[t-1] & k_1[t] = K_1 h[t-1] & v_1[t] = V_1 h[t-1] \\ q_2[t] = Q_2 h[t-2] & k_2[t] = K_2 h[t-2] & v_2[t] = V_2 h[t-2] \end{array}$$

- Each $q_i[t], k_i[t], v_i[t] \in \mathbb{R}^n$ because they are the product of an $n \times n$ matrix and an n-dimensional vector.

$$w_i[t] = q_i[t]^\top k_i[t] \quad \text{for } i = 0, 1, 2$$

- Each $w_i[t]$ is a scalar because it's the dot product of two n-dimensional vectors.

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

Input to Softargmax:

- The vector passed to the softmax function is $[w_0[t], w_1[t], w_2[t]]$, which is a vector in $\mathbb{R}^3$.

Output of Softargmax:

- The softmax function maps a vector in $\mathbb{R}^3$ to another vector in $\mathbb{R}^3$.
- Therefore, $a[t] \in \mathbb{R}^3$.

(b) (3 pts) Write out the system of equations that defines it. You may use set notation or ellipses (...) in your definition.

- Inputs $s_i[t]$:
  - $s_0[t] = x[t]$: base case.
  - $s_i[t] = h[t-i]$ for $i = 1$ to k: the previous k hidden states.
- Query, Key and Value matrices Q_i, K_i, V_i:
  - Separate weight matrices for each component i, allowing the model to learn different transformations for the current input and past hidden states.
- Queries, Keys, and Values:
  - $q_i[t]$: the query vector.
  - $k_i[t]$: the key vector.
  - $v_i[t]$: the value vector.
- Attention Weights $a[t]$:
  - Compute the unnormalized scores $w_i[t]$ using the dot product between queries and keys.
  - Apply the softmax function over the $k+1$ components.
- Hidden State $h[t]$:
  - The new hidden state is a weighted sum of the value vectors $v_i[t]$, using the attention weights $a_i[t]$.

(a) For $i = 0$ to k:

$$s_i[t] = \begin{cases} x[t], & i = 0 \\ h[t-i], & i = 1, 2, \ldots, k \end{cases}$$

$$q_i[t] = Q_i s_i[t], \quad k_i[t] = K_i s_i[t], \quad v_i[t] = V_i s_i[t]$$

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

(b) Attention Weights:

$$a[t] = \text{softmax}([w_0[t], w_1[t], \ldots, w_k[t]])$$

(c) Hidden State:

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

(c) (3 pts)

Continuing answer from previous question,

(a) For $i = 0$ to k:

$$s_i[t] = \begin{cases} x[t], & i = 0 \\ h[t-i], & i = 1, 2, \ldots, k \end{cases}$$

$$q_i[t] = Q_i s_i[t], \quad k_i[t] = K_i s_i[t], \quad v_i[t] = V_i s_i[t]$$

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

(b) Attention Weights:

$$a[t] = \text{softmax}([w_0[t], w_1[t], \ldots, w_k[t]])$$

(c) Hidden State:

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

Now here t goes from 0 to ∞,

- For each i from 0 to t, we need unique matrices Q_i, K_i, V_i.
- Total weight matrices at time t: $3 \times (t+1)$.

Over a sequence of length T, requires $3 \times (T+1)$ matrices. And for AttentionRNN(∞) this number will go to ∞.

Limiting the Number of Weight Matrices via Weight Sharing:

Full Weight Sharing: For $i = 0$ to k:

$$s_i[t] = \begin{cases} x[t], & i = 0 \\ h[t-i], & i = 1, 2, \ldots, k \end{cases}$$

- Use the same weight matrices for all positions:

$$q_i[t] = Q s_i[t]$$

$$k_i[t] = K s_i[t]$$

$$v_i[t] = V s_i[t]$$

- Total Weight Matrices Required: 3 matrices (Q, K, V) shared across all inputs and past states.

- Attention Weights:

$$a[t] = \text{softmax}([w_0[t], w_1[t], \ldots, w_k[t]])$$

- Hidden State:

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

(d) (5 pts) Suppose the loss ℓ is computed. Please write down the expression for $\frac{\partial h[t]}{\partial h[t-1]}$ for AttentionRNN(2).

The derivative $\frac{\partial h[t]}{\partial h[t-1]}$ can be decomposed into :

$$\frac{\partial h[t]}{\partial h[t-1]} = \sum_{i=0}^{2} \left(v_i[t] \frac{\partial a_i[t]}{\partial h[t-1]} + a_i[t] \frac{\partial v_i[t]}{\partial h[t-1]} \right).$$

1. **For** i = 0:

   - $v_0[t]$ and $a_0[t]$ depend only on $x[t]$, so:

$$\frac{\partial v_0[t]}{\partial h[t-1]} = 0, \quad \frac{\partial a_0[t]}{\partial h[t-1]} = 0.$$

2. **For** i = 1:

   - $v_1[t]$ and $a_1[t]$ depend on $h[t-1]$:
     - **Derivative of** $v_1[t]$:

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

     - **Derivative of** $a_1[t]$:

$$\frac{\partial a_1[t]}{\partial h[t-1]} = \sum_{k=0}^{2} \frac{\partial a_1[t]}{\partial w_k[t]} \frac{\partial w_k[t]}{\partial h[t-1]}.$$

3. **For** i = 2:

   - $v_2[t]$ and $a_2[t]$ depend on $h[t-2]$, which does not depends on $h[t-1]$:

$$\frac{\partial v_2[t]}{\partial h[t-1]} = 0, \quad \frac{\partial a_2[t]}{\partial h[t-1]} = 0.$$

Computing $\frac{\partial a_i[t]}{\partial w_k[t]}$:

The softargmax derivative is:

$$\frac{\partial a_i[t]}{\partial w_k[t]} = a_i[t](\delta_{ik} - a_k[t]),$$

where δ_{ik} is the Kronecker delta (1 if $i = k$, 0 otherwise).

Computing $\frac{\partial w_k[t]}{\partial h[t-1]}$:

1. **For** k = 0:

   - $w_0[t]$ depends only on $x[t]$:

$$\frac{\partial w_0[t]}{\partial h[t-1]} = 0.$$

2. **For** k = 1:

   - $w_1[t] = q_1[t]^\top k_1[t]$ with $q_1[t] = Q_1 h[t-1]$, $k_1[t] = K_1 h[t-1]$:

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

3. **For** k = 2:

   - $w_2[t]$ depends on $h[t-2]$, which does not depends on $h[t-1]$:

$$\frac{\partial w_2[t]}{\partial h[t-1]} = 0$$

Combining the Terms:

Putting it all together:

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

Computing $\frac{\partial a_1[t]}{\partial h[t-1]}$:

Using the chain rule:

$$\frac{\partial a_1[t]}{\partial h[t-1]} = \sum_{k=0}^{2} \frac{\partial a_1[t]}{\partial w_k[t]} \frac{\partial w_k[t]}{\partial h[t-1]}.$$

But since $\frac{\partial w_0[t]}{\partial h[t-1]} = 0$ and $\frac{\partial w_2[t]}{\partial h[t-1]} = 0$, this simplifies to:

$$\frac{\partial a_1[t]}{\partial h[t-1]} = \frac{\partial a_1[t]}{\partial w_1[t]} \frac{\partial w_1[t]}{\partial h[t-1]}.$$

Substituting the derivative of the softargmax:

$$\frac{\partial a_1[t]}{\partial w_1[t]} = a_1[t](1 - a_1[t]).$$

Therefore:

$$\frac{\partial a_1[t]}{\partial h[t-1]} = a_1[t](1 - a_1[t]) \left(Q_1^\top k_1[t] + K_1^\top q_1[t] \right).$$

Finally substituting $\frac{\partial a_1[t]}{\partial h[t-1]}$,

$$\boxed{\frac{\partial h[t]}{\partial h[t-1]} = v_1[t]a_1[t](1-a_1[t])\left(Q_1^\top k_1[t] + K_1^\top q_1[t]\right) + a_1[t]V_1.}$$

(e) (2 pts) Suppose we know the derivative $\frac{\partial h[t]}{\partial h[T]}$, and $\frac{\partial \ell}{\partial h[t]}$ for all $t > T$. Please write down the expression for $\frac{\partial \ell}{\partial h[T]}$ for AttentionRNN(k).

Understanding the Dependencies in AttentionRNN(k):

In the AttentionRNN(k) model, the hidden state at time t, $h[t]$, depends on:

- The current input $x[t]$.
- The previous k hidden states $h[t-1], h[t-2], \ldots, h[t-k]$.

This means that $h[T]$ influences $h[t]$ for $t = T+1$ to $T+k$.

Deriving the Expression for $\frac{\partial \ell}{\partial h[T]}$:

(a) **Direct Contribution at Time T:**

- The loss ℓ may have a direct dependency on $h[T]$ through the loss function at time T.
- This direct gradient is denoted as $\left.\frac{\partial \ell}{\partial h[T]}\right|_{\text{direct}}$.

(b) **Indirect Contributions from Future Time Steps:**

- For $t > T$, $h[T]$ can affect ℓ indirectly by influencing future hidden states $h[t]$.
- The indirect gradient is accumulated from these future time steps.

(c) **Total Gradient Computation:**

The total gradient $\frac{\partial \ell}{\partial h[T]}$ is the sum of the direct and indirect contributions:

$$\frac{\partial \ell}{\partial h[T]} = \left.\frac{\partial \ell}{\partial h[T]}\right|_{\text{direct}} + \sum_{t=T+1}^{T+k} \frac{\partial \ell}{\partial h[t]} \frac{\partial h[t]}{\partial h[T]}$$

4. **Considering the Dependency Window in AttentionRNN(k):**

- Since $h[t]$ depends on $h[T]$ only if $T \geq t-k$ (i.e., $h[T]$ is within the last k hidden states used at time t), we adjust the summation limits.
- For $t > T$, $h[t]$ depends on $h[T]$ only when $t \leq T+k$.
- Therefore, the summation over t is from $T+1$ to $\min(K, T+k)$.

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

1.3 Debugging loss curves (5pts)

(a) (2pts)

Sometimes gradient values becomes too high which updates the weights with greater change in wrong direction causing spikes. This happens when gradients explode due to high values of weights, higher learning rates and lower batch sizes.

(b) (1pts)

This is totally possible as we are using cross entropy loss, we start from log(4) = 1.39, now during backpropagation weights might get updated in such a way that value of $\hat{y}$ becomes large which in return gives us high loss than original.

(c) (1pts) What are some ways to fix them?

We can reduce the learning rate, clip the gradients, increase batch size to fix the issue.

(d) (1pts)

I am observing around 25 % accuracy at the initial reason is because there are total four states "Q", "R", "S" and "U". And without any training model is likely to be 25 % accurate. This is the reason we are observing the initial accuracy.

2 Implementation (50pts)