Deep Learning HW2

yg2709

February 2024

1 Theory

1.1 Convolutional Neural Networks

(a) $H = (11-5)/4+1 = 2.5$,
$W = (21-4)/4+1 = 5.25$. After rounding up, the output dimension will be 2×5.

(b) First let's consider the output dimension of each filter. $H_{out} = \frac{(H+2\times P-D\times(K-1)-1}{S}+1$. $W_{out} = \frac{(W+2\times P-D\times(K-1)-1}{S}+1$. The total output dimension will be $F \times H_{out} \times W_{out}$

(c) (i) $f_W(x) \in \mathbb{R}^{1\times 3}, f_W(x)[0][i] = \Sigma_{m=0}^{4}\Sigma_{n=0}^{2} W[0,m,n]x[m, i\times 2+n]$

(ii) $\frac{\partial f_W(x)}{\partial W} \in \mathbb{R}^{1\times 3\times 15}, \frac{\partial f_W(x)}{\partial W}[0,i,j] = x[j\%3 + 2\times i, j//3]$

(iii) $\frac{\partial f_W(x)}{\partial x} \in \mathbb{R}^{5\times 21}, \frac{\partial f_W(x)}{\partial x}[i, 7m\times 3n] = W[0,i,n] form \in 0,1,2, n \in 0,1,2$, and the rest of $\frac{\partial f_W(x)}{\partial x}$ are all 0.

(iv) $\frac{\partial l}{\partial W} \in \mathbb{R}^{1\times 15}, \frac{\partial l}{\partial W}[0,i] = \Sigma_{n=0}^{2}\frac{\partial l}{\partial f_W(x)}[0,n]x[i//3, i\%3+2\times n]$. They both have a summation from 0 to 2, and the x index has the same format of $a+2\times b$, but the total dimension is different.

1.2 Recurrent Neural Networks

1.2.1 Part 1

(a) See Fig. 1

(b) $c[t] \in \mathbb{R}^m$

(c) $\frac{\partial l}{\partial W_x} \in \mathbb{R}^{m\times n}$,
$\frac{\partial l}{\partial W_x} = ((1-c[t]) \circ \frac{\partial l}{\partial h[t]})x[t]^T$. Both forward and backward part includes two inputs: x and h

(d) The model is not subject to gradient vanishing, because through the part $c[t] \circ h[t-1]$, part of the gradient of h[t-1] doesn't go through the sigmoid function, and got preserved. However, this also makes the network more likely to explode gradient.

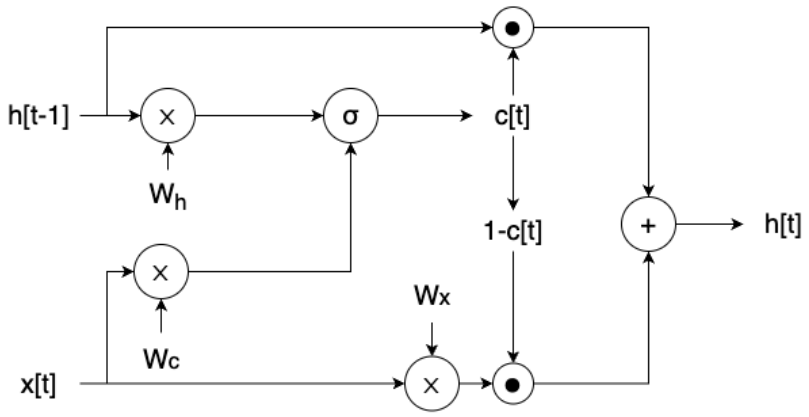

Figure 1: RNN diagram

1.2.2 Part 2

(a) See Fig. 2

(b) $a[t] \in \mathbb{R}^3$

(c) We define AttentionRNN(k) as

$$\begin{aligned} q_0[t], q_1[t], ...q_k[t] &= Q_0x[t], Q_1h[t-1], ...Q_kh[t-k] \\ k_0[t], q_1[t], ...k_k[t] &= K_0x[t], K_1h[t-1], ...K_kh[t-k] \\ v_0[t], v_1[t], ...v_k[t] &= V_0x[t], V_1h[t-1], ...V_kh[t-k] \\ w_i[t] &= q_i[t]^T k_i[t] \\ a[t] &= softargmax(w_0[t], w_1[t], ...w_k[t]) \\ h[t] &= \Sigma_{i=0}^{k} a_i[t]v_i[t] \end{aligned}$$

(d) We define AttentionRNN(∞) as

$$\begin{aligned} q_0[t], q_1[t], q_2[t], ... &= Qx[t], Qh[t-1], Qh[t-2], ... \\ k_0[t], k_1[t], k_2[t], ... &= Kx[t], Kh[t-1], Kh[t-2], ... \\ v_0[t], v_1[t], v_2[t], ... &= Vx[t], Vh[t-1], Vh[t-2], ... \\ w_i[t] &= q_i[t]^T k_i[t] \\ a[t] &= softargmax(w_0[t], w_1[t], w_2[t], ...) \\ h[t] &= \Sigma_{i=0}^{\infty} a_i[t]v_i[t] \end{aligned}$$

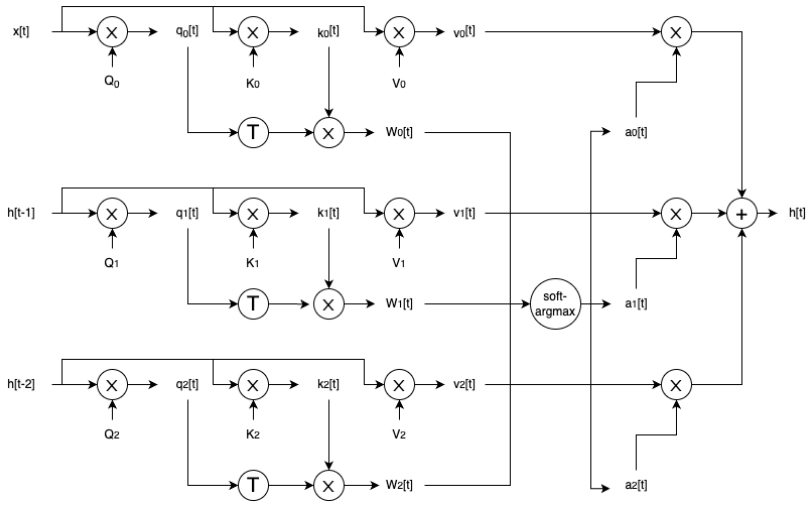

Figure 2: Attention RNN

(e)

$$\begin{aligned}
\frac{\partial h[t]}{\partial h[t-1]} &= \frac{\partial a_0[t]v_0[t] + a_1[t]v_1[t] + a_2[t]v_2[t]}{\partial h[t-1]} \\
&= \frac{\partial a_0[t]v_0[t]}{\partial h[t-1]} + \frac{\partial a_1[t]v_1[t]}{\partial h[t-1]} + \frac{\partial a_2[t]v_2[t]}{\partial h[t-1]} \\
&= \frac{\partial a_0[t]}{\partial w_1[t]}\frac{\partial w_1[t]}{\partial h[t-1]}v_0[t] + \frac{\partial a_1[t]}{\partial w_1[t]}\frac{\partial w_1[t]}{\partial h[t-1]}v_1[t] + a_1[t]V_1 + \frac{\partial a_2[t]}{\partial w_1[t]}\frac{\partial w_1[t]}{\partial h[t-1]}v_2[t] \\
&= (\frac{v_1[t]}{\Sigma_{i=0}^2 w_i[t]} - \frac{\Sigma_{i=0}^2 w_i[t]v_i[t]}{(\Sigma_{i=0}^2 w_i[t])^2})\frac{\partial w_1[t]}{\partial h[t-1]} + a_1[t]V_1 \\
&= (\frac{v_1[t]}{\Sigma_{i=0}^2 w_i[t]} - \frac{\Sigma_{i=0}^2 w_i[t]v_i[t]}{(\Sigma_{i=0}^2 w_i[t])^2})(Q_1^T k_1[t] + q_1[t]^T K_1) + a_1[t]V_1
\end{aligned}$$

(f) $\frac{\partial l}{\partial h[T]} = \Sigma_{(t>T)} \frac{\partial l}{\partial h[t]} \frac{\partial h[t]}{\partial h[T]}$

1.3 Debugging loss curves

(a) Gradient Explodes

(b) The gradient calculated during backward path is somehow extremely large, so the parameters move farther away to the optimal values than initial values, so the loss is also higher than the initial one.

(c) First, we can clip the gradient so that it will never get too big. Also, we can apply batch normalization. Besides, we can use a learning rate scheduler.

(d) The output is in a 4 sequence class Q, R, S, U, so the probability of making a correct random guess is 0.25. For cross entropy loss, the initial value will be $-(1 \times ln(0.25) + 3 \times 0 \times ln(0.25)) \approx 1.39$. Therefore loss and accuracy starts from around 1.39 and around 0.25.