DL-HW2

Jianing Zhang jz5212

Feb 15th 2024

1.1 Convolutional Neural Networks

Answer to (a)

Given an input image of dimension 11×21, the output dimension after applying a convolution with a 5×4 kernel, stride of 4, and no padding is calculated as follows:

$$\begin{aligned}
\text{output height} &= \left\lfloor \frac{\text{input height} + 2P - \text{kernel height}}{\text{stride}} \right\rfloor + 1 \\
\text{output width} &= \left\lfloor \frac{\text{input width} + 2P - \text{filter width}}{\text{stride}} \right\rfloor + 1 \\
\text{output height} &= \left\lfloor \frac{11 + 2 \cdot 0 - 5}{4} \right\rfloor + 1 \\
&= 2 \\
\text{output width} &= \left\lfloor \frac{21 - 2 \cdot 0 - 4}{4} \right\rfloor + 1 \\
&= 5
\end{aligned}$$

Therefore, the output dimension is 2×5.

Answer to (b)

Given an input of dimension $C \times H \times W$, and a convolutional layer with kernel size $K \times K$, padding P, stride S, dilation D, and F filters, the output dimensions are given by:

$$\begin{aligned}
\text{output depth(F)} &= F \\
\text{output height(H}_1\text{)} &= \left\lfloor \frac{H + 2P - D(K-1) - 1}{S} \right\rfloor + 1 \\
\text{output width(F}_1\text{)} &= \left\lfloor \frac{W + 2P - D(K-1) - 1}{S} \right\rfloor + 1
\end{aligned}$$

Therefore, the output dimension is $F \times H_1 \times W_1$.

Answer to (c)

0.1 Answer to (i)

Given an input sequence $x[n] \in \mathbb{R}^5$ with $1 \leq n \leq 7$ and a convolutional layer $f_W(x)$ with one filter of kernel size 3, stride of 2, and no dilation or padding, the output dimension can be computed as follows:
Since there is no padding (P=0) and no dilation (D=1), the formula for the output length (L) of the convolution is given by:

$$L = \left\lfloor \frac{\text{input length} - \text{kernel size}}{\text{stride}} \right\rfloor + 1 \tag{1}$$

Substituting the given values into the formula:

$$L = \left\lfloor \frac{7-3}{2} \right\rfloor + 1 = 3 \tag{2}$$

As the number of filters is 1, the output dimension of the convolutional layer $f_W(x)$ is $\mathbb{R}^{1\times 3}$.

The expression for the value of the elements of the convolutional layer output $f_W(x)$ is:

$$f_W(x)[r] = \sum_{k=1}^{5} \sum_{i=1}^{3} x \cdot W_{k,i} \cdot k[2(r-1)+i]$$

0.2 Answer to (ii)

As $W \in \mathbb{R}^{1\times 5\times 3}$,

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

The expression for the values of the derivative is

$$f_W(x)[r] = \sum_{k=1}^{5} \sum_{i=1}^{3} x \cdot W_{k,i} \cdot k[2(r-1)+i]$$

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

0.3 Answer to (iii)

The dimension of $\frac{\partial f_W(x)}{\partial x}$ is

$$\frac{\partial f_W(x)}{\partial x} \in \mathbb{R}^{(3)\times(5\times 7)}$$

The expression for the values of the derivative $\frac{\partial f_W(x)}{\partial x}$ is

$$\frac{\partial f_W(x)}{\partial x}[r, k, i] = W_{1,k,i-2(r-1)}$$

when $1 \leq i - 2(r-1) \leq 3$. And 0 otherwise.

0.4 Answer to (iv)

$$\frac{\partial l}{\partial W} \in \mathbb{R}^{1\times 5\times 3}$$

The expression for the values of the derivative is

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

The similarity is forward and backward pass in applying a convolution. And the calculation both have shared dimensions. The difference is that when do the backward pass, the stride becomes a dilation.

1.2 Recurrent Neural Networks

1.2.1 (a)

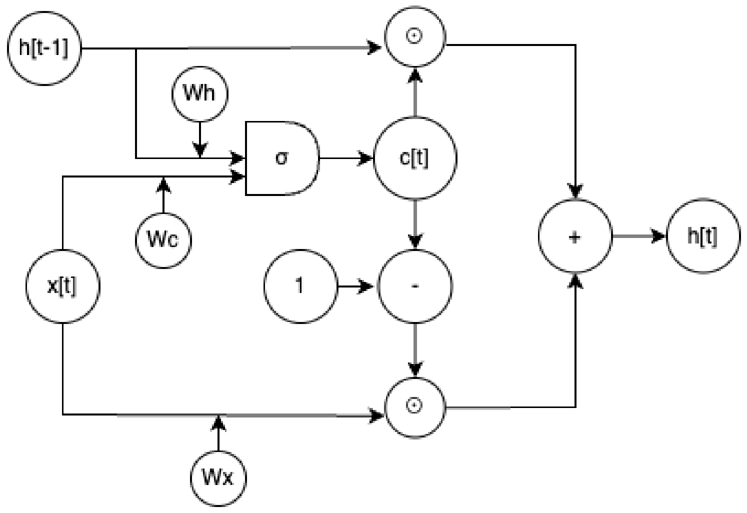

Figure 1: RNN diagram

1.2.1 (b)

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

1.2.1 (c)

$$
\begin{aligned}
\frac{\partial \ell}{\partial W_x} &= \sum_{t=1}^{K} \frac{\partial \ell}{\partial h[t]} \cdot \frac{\partial h[t]}{\partial W_x} \\
\frac{\partial h[t]}{\partial W_x} &= \frac{\partial h[t]}{\partial h[t-1]} \frac{\partial h[t-1]}{\partial W_x} \\
\frac{\partial \ell}{\partial W_x} &= \sum_{t=1}^{K} \frac{\partial \ell}{\partial h[t]} \frac{\partial h[t]}{\partial W_x} \\
&= \sum_{t=1}^{K} \frac{\partial \ell}{\partial h[t]} \left(\frac{\partial[(1-c[t]) \odot W_x x[t]]}{\partial W_x} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial h[i]}{\partial h[i-1]} \frac{\partial h[i]}{\partial W_x} \right) \right) \\
&= \sum_{t=1}^{K} \frac{\partial \ell}{\partial h[t]} \left((1-c[t]) \odot X + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial h[i]}{\partial h[i-1]} \frac{\partial h[i]}{\partial W_x} \right) \right) \\
&= \sum_{t=1}^{K} \frac{\partial \ell}{\partial h[t]} \left((1-c[t]) \odot X + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial h[i]}{\partial h[i-1]} \right) [(1-c[t]) \odot X] \right)
\end{aligned}
$$

where, $X_j = [0 \times (j-1), x[t], 0, \ldots, 0] \in \mathbb{R}^{m \times (m \times n)}$. And the dimension of $\frac{\partial \ell}{\partial W_x}$ is $\mathbb{R}^{m \times n}$, which is the same dimension as W_x.
Both of them use recursion to solve the problem.

1.2.1 (d)

The RNN network can be subject to vanishing gradients but not exploding gradients.The reason it is not subject to exploding gradients is that the state vector z_t is not subjected to recurrent matrix multiplication across timesteps. However, it is subject to vanishing gradients because, at each timestep, the state vector z_t is element-wise multiplied by the vector c_t, which contains values between 0 and 1. Multiplying by values less than one repeatedly can make the gradients become progressively smaller as they are propagated backward through time. If the gradients become too small, the weight updates during training will be negligible, which can prevent the network from learning long-range dependencies.

1.2.2(a)

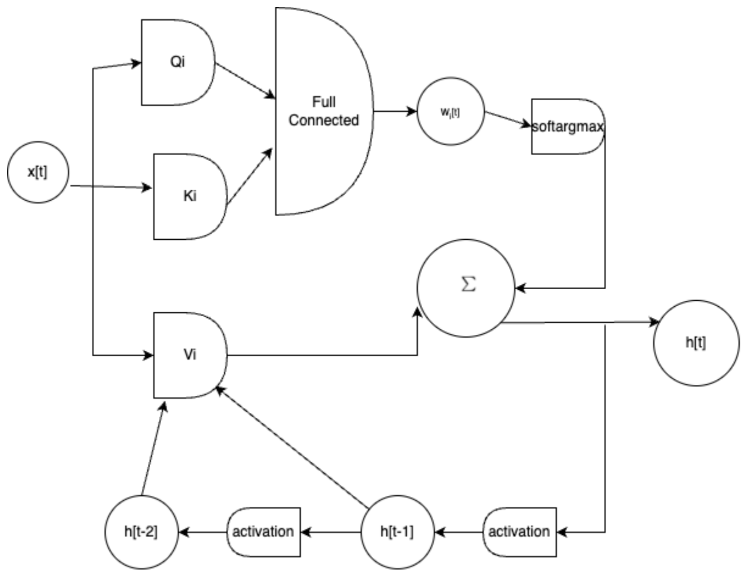

Figure 2: RNN diagram

1.2.2(b)

The dimension of $a[t]$ is $\mathbb{R}^3$.

1.2.2(c)

AttentionRNN(k)

$$\begin{aligned}
q_0[t], q_1[t], \ldots, q_k[t] &= Q_0 x[t], Q_1 h[t-1], \ldots, Q_k h[t-k] \\
k_0[t], q_1[t], \ldots, k_k[t] &= K_0 x[t], K_1 h[t-1], \ldots, K_k h[t-k] \\
v_0[t], v_1[t], \ldots, v_k[t] &= V_0 x[t], V_1 h[t-1], \ldots, V_k h[t-k] \\
w_i[t] &= q_i[t]^T k_i[t] \\
a[t] &= \text{softargmax}([w_0[t], w_1[t], \ldots, w_k[t]]) \\
h[t] &= \sum_{i=0}^{k} a_i[t] v_i[t]
\end{aligned}$$

1.2.2(d)

AttentionRNN(∞)

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

1.2.2(e)

$$\begin{aligned}
\frac{\partial h[t]}{\partial h[t-1]} &= \frac{\partial a_0[t] v_0[t]}{\partial h[t-1]} + \frac{\partial a_1[t] v_1[t]}{\partial h[t-1]} + \frac{\partial a_2[t] v_2[t]}{\partial h[t-1]} \\
&= \frac{\partial a_0[t]}{\partial h[t-1]} v_0[t] + \frac{\partial a_1[t]}{\partial h[t-1]} v_1[t] + \frac{\partial a_2[t]}{\partial h[t-1]} v_2[t]
\end{aligned}$$

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

$$= \left(\left(\frac{v_1[t]}{\sum_{i=0} w_i[t] v_i[t]} - \frac{\sum_{i=0}^{2} w_i[t] v_i[t]}{\left(\sum_{i=0}^{2} w_i[t] \right)^2} \right) \frac{\partial w_1[t]}{\partial h[t-1]} \right) - a_1[t] V_1$$

$$= \left(\frac{v_1[t]}{\sum_{i=0} w_i[t] v_i[t]} - \frac{\sum_{i=0}^{2} w_i[t] v_i[t]}{\left(\sum_{i=0}^{2} w_i[t] \right)^2} \right) (Q_1^T k_1[t] + q_1[t]^T K_1) + a_1[t] V_1$$

1.2.2(f)

$\frac{\partial \ell}{\partial h[T]} = \sum_{i=1}^{k} \frac{\partial \ell}{\partial h[T+i]} \frac{\partial h[T+i]}{\partial h[T]}$.

1.3 Debugging loss curves

Answer to 1

The spike on the left is because of the high learning rate, which makes the updates too large and overshoots the optimal values. Also, maybe the batch size is a little bit small so that the gradient estimates might be noisy, leading to fluctuating loss values.

Answer to 2

If the learning rate is too high, the updates to the model's weights can be so large that they actually increase the loss rather than decrease it.

Answer to 3

We have several ways to solve this problem. We can decrease the learning rate, using gradient clipping to prevent the gradients from becoming too large, increase the batch size for a more stable gradient estimate or implementing learning rate schedules that decrease the learning rate over time or adaptively adjust the learning rate based on the training process.

Answer to 4

As there are 4 possibilities Q, R, S, U, we are guessing ramdomly with the possibility $\frac{1}{4}$, and then we take natural log so that $log(4) = 1.3$.