Deep Learning HW3

Liam Carpenter

October 2021

1 Convolutional Neural Networks 1.1

a)

Input $X \in R^{10\times 11}$ Rows Out: $\frac{10-2-1}{2} = 4.5$ which pytorch will round down to 4. Columns Out: $\frac{11-2-1}{2} = 5$ So our final output will be $\in R^{4\times 5}$

b)

Input $B \in R^{C\times H\times W}$. Convolution defined by: Kernel $K \times K$, Padding P, Stride S, Dilation D, Filters F.

Then output will be
Channels Out: F
Rows Out: $\frac{H+2P-DK+D-1}{S} + 1$
Columns Out:$\frac{W+2P-DK+D-1}{S} + 1$

c)

i)

$f_W(x) \in R^{1\times 1\times 2}$ Along each of the input dimensions we preform the convolution operation,

$$f_W(x) = \sum_{r=1}^{2} (x * W)[r]$$

Proceeding element-wise and expanding this out,

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

ii)

The derivative of this operation with respect to a specific weight is best seen through the full expansion of elements of $f(x)$.

$$f(x)[1,1,1] = x[1,1,1]W[1,1,1] + x[1,1,2]W[1,1,2] + x[1,1,3]W[1,1,3] +$$

$$x[1,2,1]W[1,2,1] + x[1,2,2]W[1,2,2] + x[1,2,3]W[1,2,3]$$

$$f(x)[1,1,2] = x[1,1,4]W[1,1,1] + x[1,1,5]W[1,1,2] + x[1,1,6]W[1,1,3] +$$

$$x[1,2,4]W[1,2,1] + +x[1,2,5]W[1,2,2] + x[1,2,6]W[1,2,3]$$

From this it is easy to see that the derivative with respect to a weight $W[r,q,z]$ will take the x that multiplies that weight. We can write this formally using our above formula as,

$$\frac{\partial f_W(x)_{1,1,p}}{\partial W_{1,q,z}} = \begin{cases} x[1,r,2p-2+m] & r=q, m=z \\ 0 \quad \text{otherwise} \end{cases}$$

Then the gradient $\frac{\partial f_W(x)}{\partial W} \in R^{2\times 3\times 2}$.

iii)

From the expansions above we can see that $\frac{\partial f_W(x)}{\partial x}$ can be found using a similar scheme. For a choice of $f_W(x)[1,1,i]$ and $x[1,t,p]$ the derivative will select the weight in the expanded expression that multiplies $x[1,t,p]$. Formally we write

$$\frac{\partial f_W(x)_{1,1,p}}{\partial x_{1,d,s}} = \begin{cases} W[1,r,m] & r=d, m=s-2p+2 \\ 0 \quad \text{otherwise} \end{cases}$$

Then $\frac{\partial f_W(x)}{\partial x} \in R^{2\times 5\times 2}$

iv)

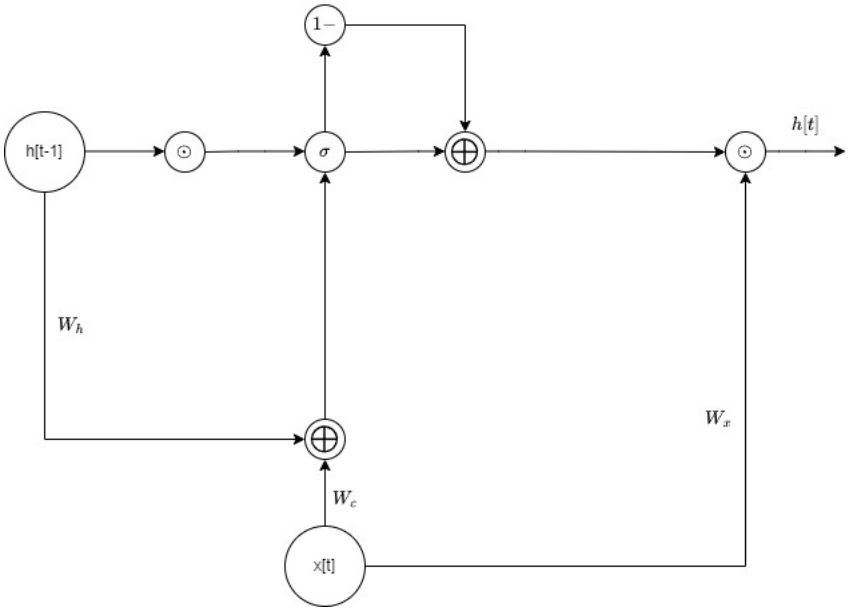
Given $\frac{\partial \ell}{\partial f_W(x)}$ we can find the gradient with respect to the input weights as $\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial f_W(x)} \frac{\partial f_W(x)}{\partial W}$

$$\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial f_W(x)_1} \frac{\partial f_W(x)}{\partial W_1} + \frac{\partial \ell}{\partial f_W(x)_1} \frac{\partial f_W(x)}{\partial W_2}$$

Which gives the appropriate dimensions for our update of $\frac{\partial \ell}{\partial W} \in R^{1\times 2\times 3}$

1.2

a)



b)

$c[t] \in R^m$

c)

The expression for error gradients with respect to the weights in this example involve a not so simple use of the chain rule.

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

This second term on the right has multiple dependencies on W_x. Liberal use of the product rule will help us get through it together.

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

Collecting some terms gives us,

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

G This gives our final recursive formula for the gradients of the last with respect to the weight,

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

Also note that

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

For the back and the forward bass the RNN induces dependencies on the layer that comes before it explicitly through he incorporation of the $h[t-1]$ but also in a "soft" way through the sigmoidaly activated c[t](and its gradient).

d)

Because when we backpropogate the gradients we "unroll" the network we end up creating a very deep network in terms of the back propogation. We know that very deep neural networks are those at the highest risk for vanishing and exploding gradients. Similarly the networks are repeating the same product at in the backpropogation for each layer. further contributing to the potential for vanishing and exploding gradients.