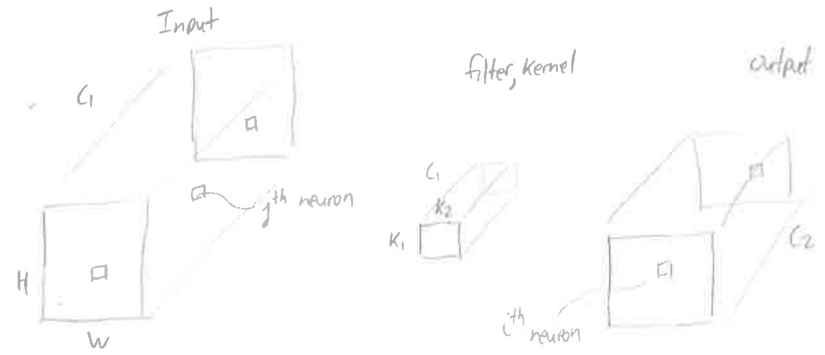
Given

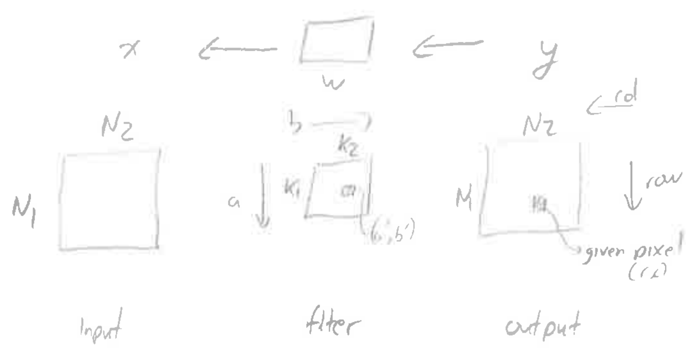


C_1 can be depth for example 3 for an image with RGB

$C_2 \Rightarrow$ # of Kernels/filters

Let's assume that C_1 + C_2 are 1 to prevent subscripts but we can extend this later

Forward propagation is easy, its just convolution of the input + filters



<u>Note</u>

c+r are going from $0 \to N_2$, $0 \to N_1$
b,a are going from $0 \to K$, $0 \to K_2$

; x, y are the same because we are assuming that we have added appropriate padding

$\frac{\partial L}{\partial x} = ?$ $\frac{\partial L}{\partial w} = ?$ $\frac{\partial L}{\partial y} \to$ is given b/c its the output of the network

We know that

$$y[r,c] = \sum_{a=0}^{K_1} \sum_{b=0}^{K_2} x[r+a, c+b]\, w[a,b] \quad (1)$$

$\leftarrow$ this is just $x * w$ centered @ pixel r,c

This double summation is just a dot product of 2 matrices

$$[\ \vec{x}_{vec}\ ] \begin{bmatrix} \vec{w}_{vec} \end{bmatrix}$$

$\Rightarrow$ to get a value @ y you can vectorize the input + weights + perform a dot product

In some libraries this is what they do. They vectorize the entire input space + vectorize the $\vec{w}$ + perform the dot product

Now $\frac{\partial L}{\partial y}$ is a matrix + can be indexed to $\frac{\partial L}{\partial y(r,c)}$ @ a particular output pixel (r,c)

If we specifically look @ $\frac{\partial L}{\partial w(a',b')}$

Unlike traditional ANN if you change 1 weight it only affects 1 neuron, by changing 1 parameter in the filter you essentially affect the entire output image. This is because of the convolution operation over the entire input image to generate output image

Let us try + figure of the first change in back prop

$$\boxed{\frac{\partial L}{\partial w[a',b']}} = \sum_{r=0}^{N_1-1} \sum_{c=0}^{N_2-1} \frac{\partial L}{\partial y[r,c]} \frac{\partial y[r,c]}{w[a',b']}$$

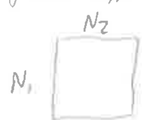
by looking @ eq'n ① $\frac{\partial y[r,c]}{w[a',b']}$ is simply $x[r+a', c+b']$

$$\Rightarrow \underbrace{\sum_{r=0}^{N_1-1} \sum_{c=0}^{N_2-1}} \frac{\partial L}{\partial y[r,c]} \; x[r+a', c+b']$$

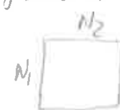
⇒ This eq'n should look familiar as it is simply the convolution eq'n w/an offset of a' + b' to the input image.

These 2 summations are as a result of parameter sharing. Meaning each output neuron is sharing the same parameter as the other output neurons. This is b/c each output is the product of input times the filter/kernel. This is also the reason why you collect all the data from all of the outputs in this summation.

This is a matrix of size $N_1 \times N_2$



This is also a matrix of size $N_1 \times N_2$



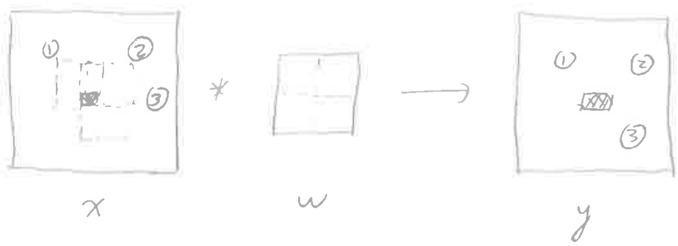
$$\boxed{\therefore \frac{\partial L}{\partial w} \text{ is simply a convolution of the loss + input values w/an offset of } a' + b'}$$

let us figure out the second change in back-prop

$$\boxed{\frac{\partial L}{\partial x[r',c']}}$$

So, what does the pixel really affect.

Given pixel$(r',c') \rightarrow$ ▨



@ important points:

① → the first time r',c' plays a role

③ → the last time r',c' plays a role

② → somewhere in b/w

Due to the nature of shifting around the centre of the kernel the input affects the neighbouring pixels of r',c' in the y matrix. The size of affect is determined by the size of the w matrix.

Looking at this we can see that $x[r',c']$ affects the output pixels as

these are the pixels affected & they are governed by $(r'-K_1+1, c'-K_2+1)$

→ this means that we only care about affected pixels "p"

$$\frac{\partial L}{\partial x[r',c']} = \sum_p \frac{\partial L}{\partial y(p)} \frac{\partial y[p]}{\partial x[r',c']}$$

$$= \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} \underbrace{\left(\frac{\partial L}{\partial y[r'-a,c'-b]}\right)}_{\text{like the previous calculations these are just matrices}} \underbrace{\frac{\partial y[p]}{\partial x[r',c']}}_{\text{This is what we have to figure out}}$$

Recall for vanilla backprop $y = wx$ the $\frac{\partial y}{\partial x} = w$. This is similar as $y = x * w$ $\therefore \frac{\partial y}{\partial x} = w$

Alternatively, if you look @ eq'n ① on the first page it is simply $w[a',b']$

$$y[r'-a, c'-b] = \sum_{a'=0}^{K_1-1} \sum_{b'=0}^{K_2-1} x[r'-a+a', c'-b+b'] \, w[a',b']$$ * change of variables

$$\Rightarrow \frac{\partial y[r'-a, c'-b]}{\partial x[r',c']} = w[a',b']$$

$$\therefore \frac{\partial L}{\partial x[r',c']} = \underbrace{\sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} \frac{\partial L}{\partial y[r'-a,c'-b]} \, w[a,b]}_{\text{This is not a convolution but rather a crosscorrelation. So if we flip } w \text{ on horizontal + vertical}}$$

look closely @ the subscripts of the y. It's $r'-a$ instead of $r+a$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} * w^{flip}$$

Analytical Reasoning w/ example

Consider a 1D case $Input = [a, b, c, d, e]$ $w = [x, y]$

Apply convolution we get $I * w = Output = O$ w/ zero padding we get

$O = [(0x + ay), (ax + by), (bx + cy), (cx + dy), (dx + ey), (ex + 0y)]$

Now taking each partial derivatives w/ respect to entire input space we get

$\frac{\partial O}{\partial a} = [y, x, 0, 0, 0, 0]$ $\frac{\partial O}{\partial d} = [0, 0, 0, y, x, 0]$

$\frac{\partial O}{\partial b} = [0, y, x, 0, 0, 0]$ $\frac{\partial O}{\partial e} = [0, 0, 0, 0, y, x]$

$\frac{\partial O}{\partial c} = [0, 0, y, x, 0, 0]$

Observe that the Kernel is flipped when you take the derivative wrt each input pixel.

Now looking @ the loss function or error $E(o)$

Consider $\frac{\partial E}{\partial b} = \frac{\partial E}{\partial O}^T \frac{\partial O}{\partial b}$

there is a transpose to keep the dimensions when performing the dot product

This is found above

This is just a vector of loss function similar to the input above
Let this vector be $E = [f, g, h, i, j]$

$= [f, g, h, i, j] \cdot [0, y, x, 0, 0]$

$\frac{\partial E}{\partial b} = gy + hx$

As you can see this is also the flipped version of the kernel

We can extend this to 2D but we flip in both horizontal + vertical axis

A more detailed explanation of 2D can be seen below,

Given → Input matrix I

→ Kernel/filter K w/size $n \times n$ where $n = 3, 5, 7$

→ Output matrix O w/loss function given by $E(O)$ such that a matrix $\frac{\partial E}{\partial O}$ exists then

$$O_{xy} = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} I_{(x+u,\, y+u)} \cdot K_{(u,v)} \quad ①$$

$$\frac{\partial E}{\partial I_{(i,j)}} = \sum_{xy} \frac{\partial E}{\partial O_{xy}} \frac{\partial O_{(x,y)}}{\partial I_{(i,j)}} \quad ②$$

Substituting ① into ② we get

$$\frac{\partial E}{\partial I_{(i,j)}} = \sum_{xy} \frac{\partial E}{\partial O_{(x,y)}} \frac{\partial \left(\sum_{u=0}^{n-1} \sum_{v=0}^{n-1} I_{(x+u,\, y+u)} \cdot K_{(u,v)} \right)}{\partial I_{(i,j)}}$$

The dual summation can be moved outside the partial derivatives + so does $K(u,v)$ b/c there are no "I" terms in their

$$\frac{\partial E}{\partial I_{(i,j)}} = \sum_{xy} \frac{\partial E}{\partial O_{(x,y)}} \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} K_{(u,v)} \frac{\partial I_{(x+u,\, y+u)}}{\partial I_{(i,j)}}$$

Now if you look @ the last term + using the previous example I did this holds only true for specific indices + the rest are zeros. These are where the x, y are present in the output.

$$\frac{\partial E}{\partial I_{(i,j)}} = \sum_{xy} \frac{\partial E}{\partial O_{(xy)}} \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} K_{(u,v)} \cdot 1\,(x+u=i,\ y+v=j)$$

From the above we can say that this is only true for values

$x+u=i$ for some u w/some range $[0, n-1]$

$y+u=i$ for some v w/some range $[0, n-1]$

Lets try + combine the summation so we isolate x, y in terms of u, v + find the range

$x = (i-u)$ for u $[0, n-1]$

This means that x is in the range $[i-0,\ i-(n-1)]$

$[i,\ i-n+1]$

This is the same as y — // — // — // — $[j,\ j-n+1]$

We can then imply $u = i - x$

$+\ v = j - y$

This means we can simplify this to

$$\frac{\partial E}{\partial I_{(i,j)}} = \sum_{x=i-n+1}^{i} \sum_{y=j-n+1}^{j} \frac{\partial E}{O_{(x,y)}} K_{(u,v)}$$

b/c $x+u=i$, $y+u=j$ | $u=i-x$, $v=j-y$

$$\frac{\partial E}{\partial I_{(i,j)}} = \sum_{x=i-n+1}^{i} \sum_{y=j-n+1}^{j} \frac{\partial E}{O_{(x,y)}} K(i-x, j-y)$$

this means $(n-1-x', n-1-y')$

Now we set $x' = x-i+n-1$ $\quad y' = y-j+n-1$

We want this to be equal to ϕ b/c the derivative of linear function is ϕ

$$\frac{\partial E}{\partial I_{(i,j)}} = \sum_{x'=0}^{n-1} \sum_{y'=0}^{n-1} \frac{\partial E}{O_{(x'+i-n+1,\, y'+j-n+1)}} K(n-1-x', n-1-y')$$

This decreasing x' & y' indicates that the kernel is flipped on x & y axis