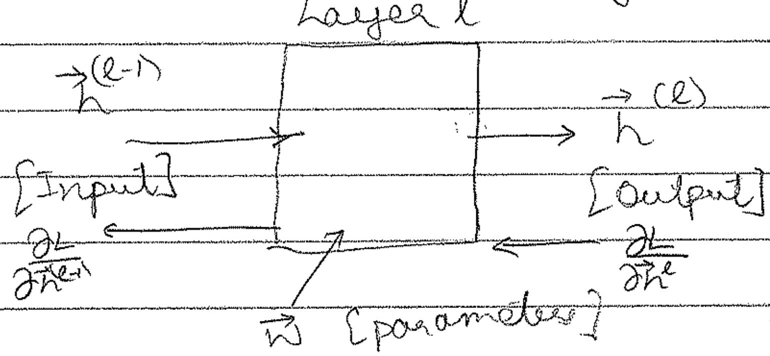
9/03/15

MLPs + CNNs

1) Recall from last time, key abstraction:



$$\vec{h}^{(l)} = g(\vec{h}^{l-1}, \vec{w})$$

Loss $L = f(\vec{h}^{l})$

Abstractly: (a) $$\frac{\partial L}{\partial \vec{in}} = \frac{\partial L}{\partial \vec{out}} \cdot \frac{\partial \vec{out}}{\partial \vec{in}}$$

(b) $$\frac{\partial L}{\partial \vec{w}} = \frac{\partial L}{\partial \vec{out}} \cdot \frac{\partial \vec{out}}{\partial \vec{in}}$$

[A bit more]
Concretely:

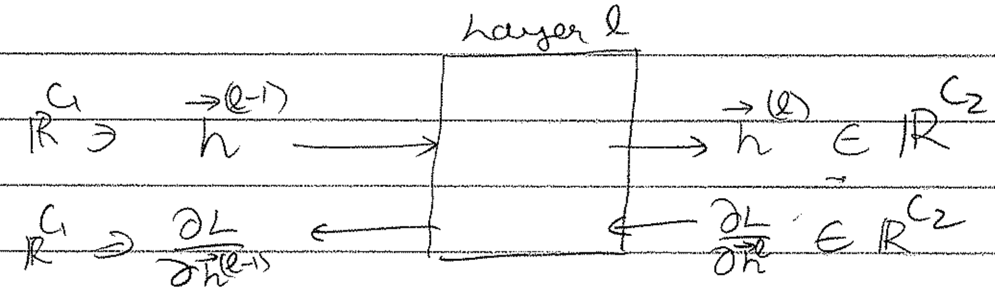
(a) $$\frac{\partial L}{\partial \vec{h}^{(l-1)}} = \frac{\partial L}{\partial \vec{h}^{l}} \times \frac{\partial \vec{h}^{l}}{\partial \vec{h}^{l-1}}$$

$$[\quad]_{1\times|\vec{h}^{l-1}|} \quad [\quad]_{1\times|\vec{h}^{l}|} \quad [\quad]_{|\vec{h}^{l}|\times|\vec{h}^{l-1}|}$$

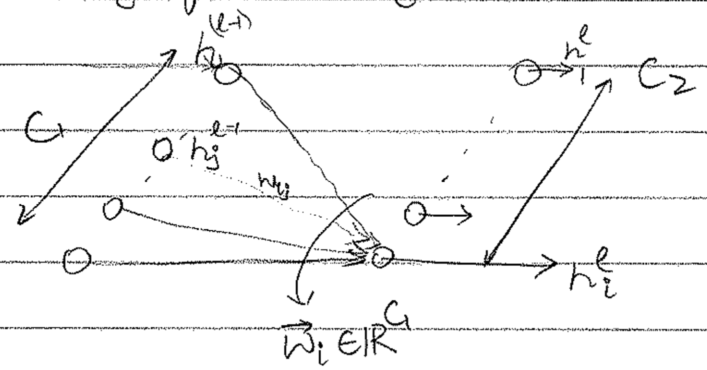
(b) $$\frac{\partial L}{\partial \vec{w}} = \frac{\partial L}{\partial \vec{h}^{(l)}} \times \frac{\partial \vec{h}^{l}}{\partial \vec{w}}$$

$$[\quad]_{1\times|\vec{w}|} \quad [\quad]_{1\times|\vec{h}^{l}|} \quad [\quad]_{|\vec{h}^{l}|\times|\vec{w}|}$$

② Multi-Layer Perceptrons / Fully-Connected Layer / Inner-Product Layer



Another way of drawing this



[Why draw in 3D? You'll see. Wait for CNNs!]

Forward-Pass:

$$h_i^l = \sum_{j=1}^{C_1} w_{ij} h_j^{(l-1)}$$

$$= \vec{w}_i^T \vec{h}^{(l-1)}$$

Let's vectorize some more

$$\vec{h}^l = W^{(l)} \vec{h}^{(l-1)}$$

$$\begin{bmatrix} h_1^l \\ \vdots \\ \vdots \\ h_{C_2}^l \end{bmatrix} = \begin{bmatrix} \leftarrow w_1^T \rightarrow \\ \vdots \\ \leftarrow w_i^T \rightarrow \\ \vdots \\ \leftarrow w_{C_2} \rightarrow \end{bmatrix} \begin{bmatrix} h_1^{l-1} \\ \vdots \\ \vdots \\ h_{C_1}^{l-1} \end{bmatrix}$$

So all of F-PASS = 1 matrix-mult

Why did we write this as matrix mult?

① Because it's cooler!

② Gradients, my dear Watson, Gradients...

$$\vec{h}^{(\ell)} = W^{(\ell)} \vec{h}^{(\ell-1)}$$

$$\Rightarrow \quad \frac{\partial \vec{h}^{(\ell)}}{\partial \vec{h}^{\ell-1}} = W^{(\ell)}$$

[Now, isn't that beautiful?]

$$\frac{\partial \vec{out}}{\partial \vec{in}}$$

What about $\dfrac{\partial\, \vec{out}}{\partial \vec{w}}$?

Simple $\quad h_i^{\ell} = \vec{w}_i^{\,T} \vec{h}^{\ell-1}$

$$\Rightarrow \quad \frac{\partial h_i^{\ell}}{\partial \vec{w}_i} = \vec{h}^{(\ell-1)\,T}$$

Now put it all together:

ⓐ $$\frac{\partial L}{\partial \vec{h}^{\ell-1}} = \frac{\partial L}{\partial \vec{h}^{\ell}} \frac{\partial \vec{h}^{\ell}}{\partial \vec{h}^{\ell-1}} = \boxed{\frac{\partial L}{\partial \vec{h}^{\ell}} \times W^{(\ell)}}$$ Very nice

So B-PASS (a) $\equiv$ 1 matrix mult

Left multiply $\equiv$ gradient
Right multiply $\equiv$ output

ⓑ $$\frac{\partial L}{\partial \vec{w}_i} = \frac{\partial L}{\partial h_i^{\ell}} \frac{\partial h_i^{(\ell)}}{\partial \vec{w}_i} = \underbrace{\frac{\partial L}{\partial h_i}}_{\text{scalar}} \times \vec{h}^{(\ell-1)\,T}$$

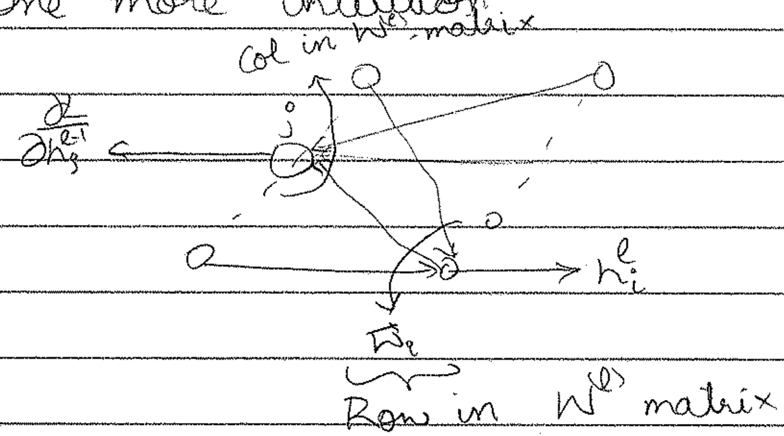
$[\quad\quad]_{1\times C_1}$

So $W^{(l)} = W^{(l)} - \eta \frac{\partial L}{\partial W^{(l)}}$

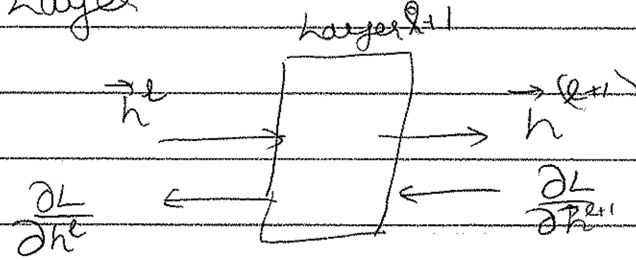
where

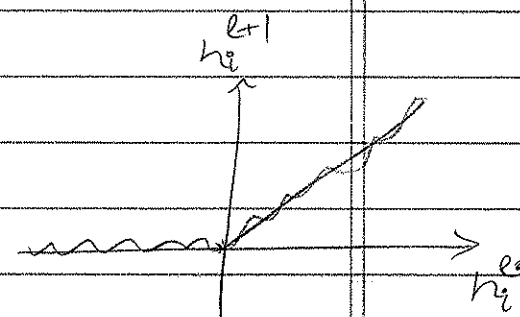
$$\frac{\partial L}{\partial W^{(l)}} = \begin{bmatrix} \longleftarrow \frac{\partial L}{\partial h^l_1} \cdot h^{(l-1)^T} \longrightarrow \\ \vdots \\ \longleftarrow \frac{\partial L}{\partial h^l_{c_2}} \cdot h^{(l-1)^T} \longrightarrow \end{bmatrix}$$

One more intuition



③ ReLU Layer





F-PASS $\quad h^{(l+1)}_i = \max\{0, h^l_i\}$

B-PASS [No params] $\quad \frac{\partial h^{l+1}_i}{\partial h^l_i} = \begin{cases} +1 & \text{if } h^l_i \geq 0 \\ 0 & \text{else} \end{cases}$

$$\Rightarrow \frac{\partial L}{\partial h^l_i} = \frac{\partial L}{\partial h^{l+1}_i} \times \mathbb{1}\{h^l_i \geq 0\}$$

(4) Recall: Convolutions! [Yes, that nightmare from signalprocessing class]

→ Let's start with pure math
then move to
math → CS → programming

Pure Math → Let $y(t)$, $x(t)$, $w(t)$ be continuous-time signals

1-D convolution $y(t) = (x * w)(t)$

$$\Rightarrow \quad y(t) = \int_{a=-\infty}^{\infty} x(a)\, \underbrace{w(t-a)}\, da$$

Intuition: to compute $y(t)$

→ Flip: $w(a) \rightarrow w(-a)$
[Why? ∵ math]
~~We won't care in ML DL~~

→ Shift by t: $w(-a) \rightarrow w(-(a-t))$

→ Compute area of product $\int_{a=-\infty}^{\infty} x(a)\, w(-(a-t))\, da$

Rinse & Repeat $\forall t$

BTW: $y(t) = \int x(a)\, w(t-a)\, da = \int x(t-a)\, w(a)\, da$
[commutative]

Similarly 2D convolution

$$y(t_1, t_2) = \int_{a=-\infty}^{\infty} \int_{b=-\infty}^{\infty} x(t_1, t_2)\, w(a-t_1, b-t_2)\, da\, db$$

Now, lets move to CS → There is no infinite precision! Let's go discrete

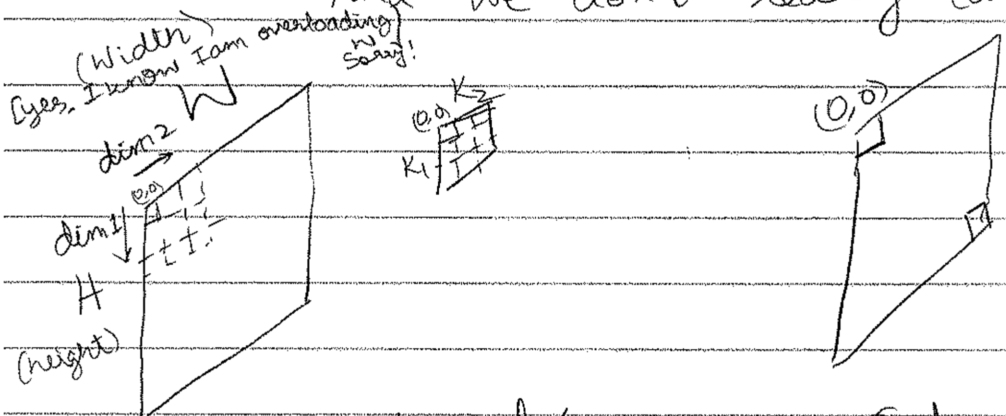
$$y[r, c] = \sum_{a=-\infty}^{\infty} \sum_{b=-\infty}^{\infty} x[r, c]\, w[a-r, b-c]$$

No inf memory either. Has to be finite matrices

$$y[r, c] = \sum_{a=-\frac{H+1}{2}}^{\frac{H+1}{2}} \sum_{b=-\frac{W+1}{2}}^{\frac{W+1}{2}} x[r, c]\, w[a-r, b-c]$$

Let's move to programming → Arrays don't index with negatives!

And we don't really care about commutivity!



Input — Kernel/Filter/Weight — Output [size depends on 'valid' or 'same' conv]

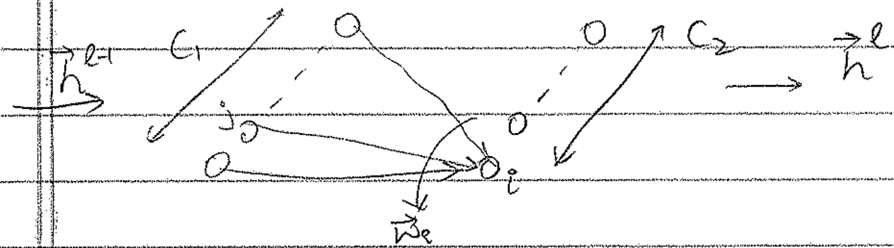
What we actually implement:

row, col

$$y[r, c] = \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} x[r+a, c+b]\, w[a, b]$$

5 CNNs / ConvNets / Convolutional Neural Nets

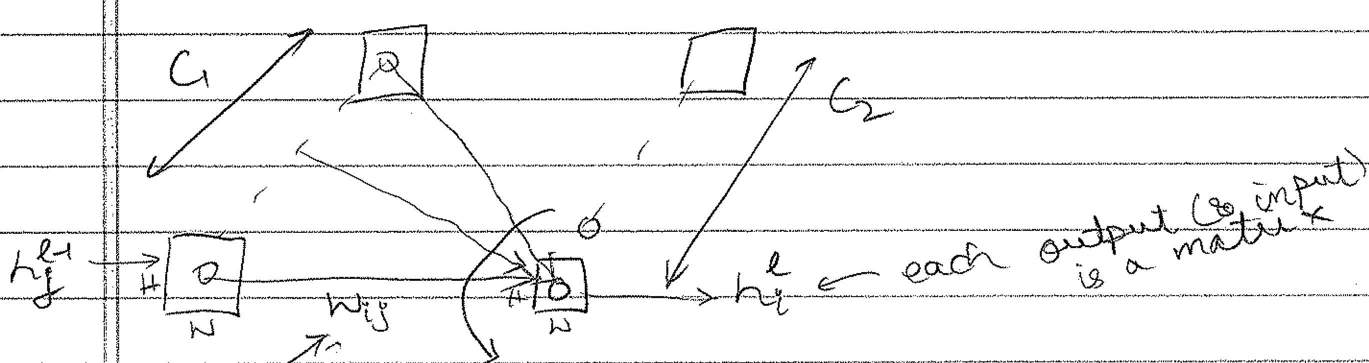
→ Recall MLP



$$h_i^l = \sum_{j=1}^{C_1} w_{ij}\, h_j^{(l-1)}$$

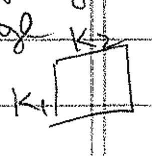
scalar product

→ CNN [Make each neuron a 2D matrix]

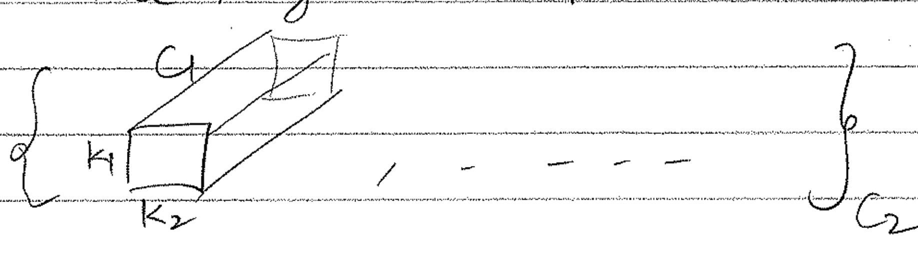


← each output (& input) is a matrix

Now a matrix / filter on every edge



All weights at output unit i



CNN - F-Pass

Compare:

scalar product

$$\text{MLP}: \quad h_i^{l} = \sum_{j=1}^{C_1} W_{ij}^{(l)} \cdot h_j^{l-1}$$

scalar

matrix

$$\text{CNN}: \quad h_i^{l} = \sum_{j=1}^{C_1} W_{ij}^{(l)} * h_j^{l-1}$$

2D conv

$$= \underbrace{\boxed{\sum_{j=1}^{C_1}}}_{\text{Input Channels}} \underbrace{\boxed{\sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1}}}_{\text{2D conv}} h_j^{(l-1)}[r+a, c+b]\, W_{ij}^{(l)}[a,b]$$

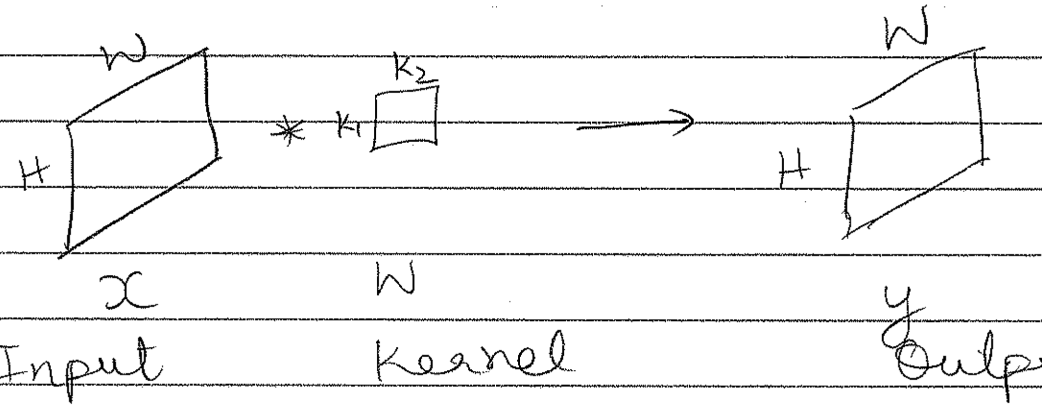
CNN - B-Pass

→ This is going to be a notation nightmare

→ So let's simplify thing a bit

→ Say $C_1 = 1$ $C_2 = 1$ [We can always generalize its just an extra sum]

→ And let's not over-load h



[assume same size for simplicity]