1D case

Convolution

NOTE: This is not predicting future. We are trying to reduce noise we have

Lets say we want to know where a spaceship is given that we know some info that is noisy

(9.28.37)
(9.28.38)
⋮
(9.29) → to find

→ we have these readings too (Lets say microseconds)

Lets consider that the only measurement device is noisy

To find the true current position

now lets find a weighted average of all the previous 60 seconds

So we have $x(a)$, $\omega(t-a)$ @ $t = 9.29$, $a = 9.28.37$

9.29

$$s(t) = \int x(a)\,\omega(t-a)\,da$$

→ time is 't' & your weight $w(t-1)$ will be the largest and then $w(t-2)$, $w(t-3)$ etc...

→ $t-a$ is different for all 'a'.

so you can treat ω as a probability distribution in order to get an "average"

$$s(t) = (x * \omega)\,t$$

↙ x convolved

$$(x-\omega)(t) = \int_{t_1}^{t_2} x(a)\,\omega(t-a)\,da$$

this is very similar to a convolution formula

$x(a)$ will be some kind of function which we don't know if continuous

So since measurements are discrete

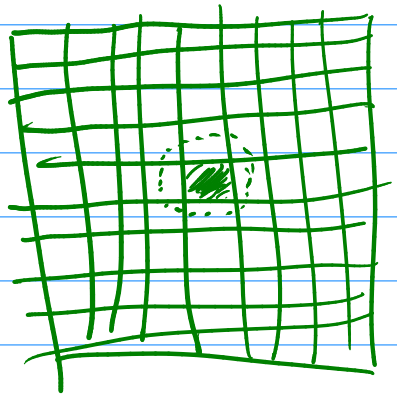
we get $\sum_{a=t_1}^{t_2} x(a)\, w(t-a)$

$$= \sum_{a=9.28.01}^{9.29} x(a)\, w(t-a)$$

this way we can reduce the effect of noise

We are not predicting future!

2D case



→ for edge detection

So we take a linear combination of nearby pixels

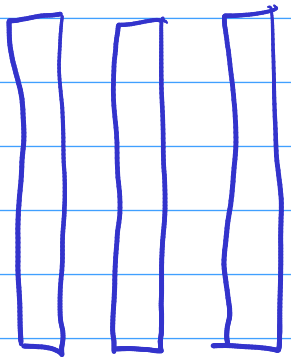
→ here, the linear combination expression will be the same for all pixels

$$S(t) = (x * w)\,t = \sum_{a} x(a)\, w(t-a)$$

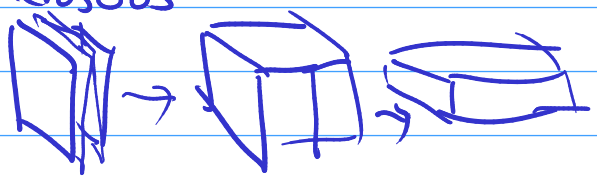
$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)\, k(i-m, j-n)$$

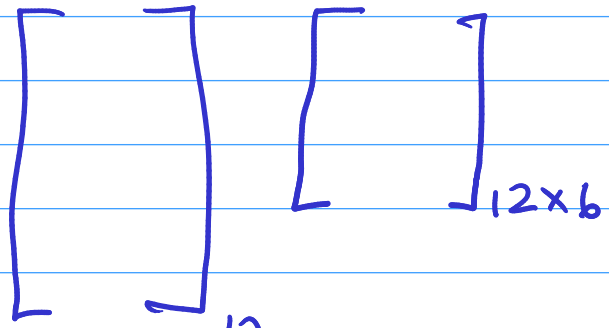
$$S(i,j) = (I * k)(i,j) = \sum_{m}\sum_{n} I(i+m, j+n)\, k(m,n)$$

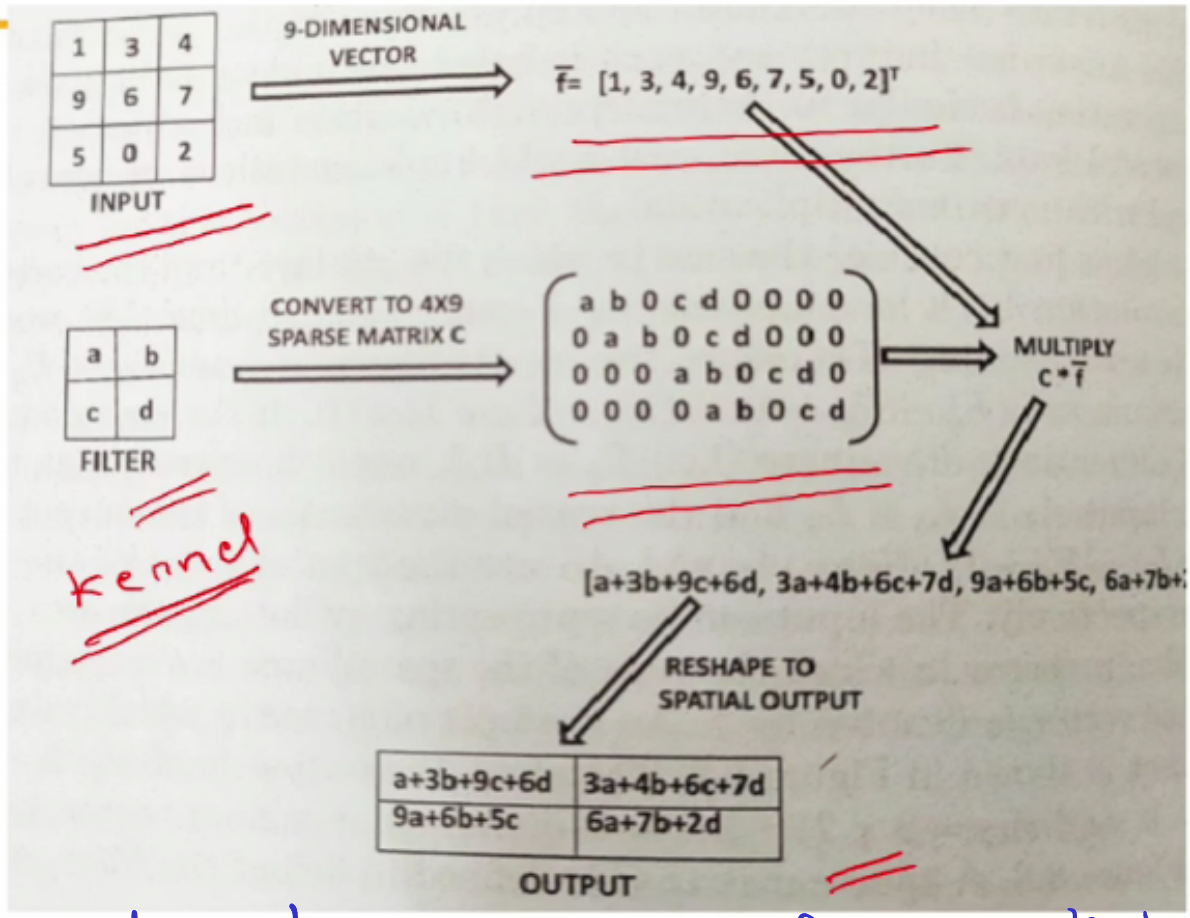
this is more a convolution w.r.t ω



You can give higher dimensional input as tensors







kernel

during backprop, mask the weights that are zero

some weights have to be the same

So we minimize loss function

$\min(L)$ subject to some weights being same

This is a very complex optimization problem (?)

So lets say $\boxed{\omega_{11} = \omega_{22} = \omega_{34} = \omega_{44} = \omega_1}$

$$\frac{\partial L}{\partial \omega_1} = \sum \left(\frac{\partial L}{\partial \omega_{11}} \cdot \frac{d\omega_{11}}{\partial \omega_1} \right)$$

$$= \frac{\partial L}{\partial \omega_{11}} \cdot \frac{\partial \omega_{11}}{\partial \omega_1} + \frac{\partial L}{\partial \omega_{22}} \frac{\partial \omega_{22}}{\partial \omega_1}$$

$$+ \frac{\partial L}{\partial \omega_{34}} \cdot \frac{\partial \omega_{34}}{\partial \omega_1} + \frac{\partial L}{\partial \omega_{44}} \cdot \frac{\partial \omega_{44}}{\partial \omega_1}$$

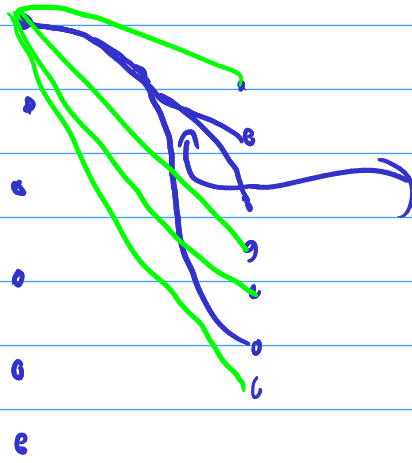
but $\omega_{11} = \omega_1 \Rightarrow \frac{\partial \omega_{11}}{\partial \omega_1} = 1$

$$\Rightarrow \frac{\partial L}{\partial \omega_1} = \frac{\partial L}{\partial \omega_{11}} + \frac{\partial L}{\partial \omega_{22}} + \frac{\partial L}{\partial \omega_{34}} + \frac{\partial L}{\partial \omega_{44}}$$

$$\omega_1^{(2)} = \omega_1^{(1)} - \eta \left. \frac{\partial L}{\partial \omega_1} \right|_{\omega_1 = \omega_1^{(1)}}$$

This way all of them will be updated the same way! LMAO

This is the concept of weight sharing



→ Shared weights, they get updated same way

→ In this case, by expanding it to a sparse matrix we increase space complexity and reduce time complexity

→ Another way is to just aggregate over a sliding window, which is time intensive but less space

But internally we dont even actually store as sparse matrix, so most implementations choose sparse conversion

A computational graph does the trick as well

weights as nodes, and multiple edges from weights ⇒ weight sharing