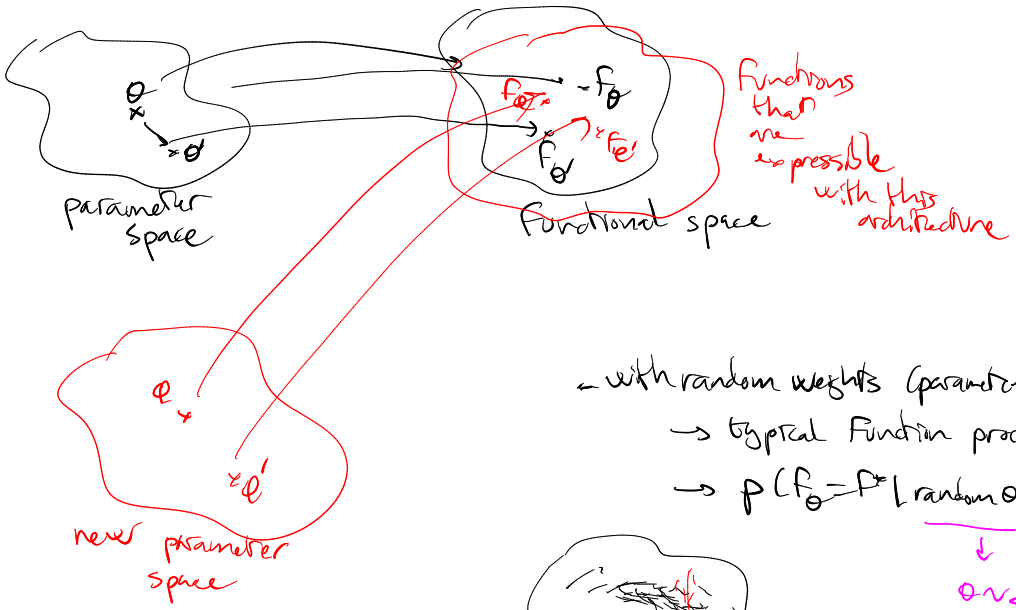
Architectures

I Architectures = prior on the function space

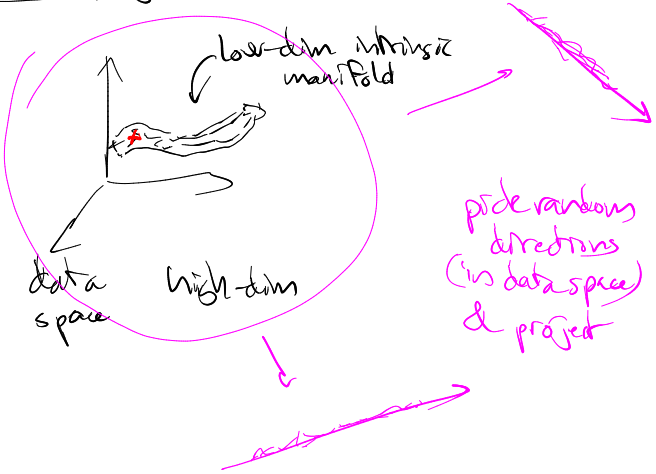
Change of paradigm:
- classical ML: design features by hand
- deep DL: meta-design of features → design architectures able to express / learn features

An architecture:



- with random weights (parameters),
  → typical function produced?
  → $p(f_\theta = f^* \mid \text{random } \theta)$
  ↓
  over θ

Random projections



pick random directions (in data space) & project

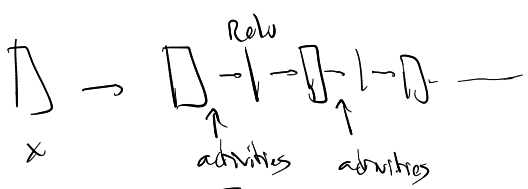
to m directions

$x \in \mathbb{R}^{1000} \longmapsto \text{Proj}(x) \in \mathbb{R}^m$

dim reduction → apply classical ML tool

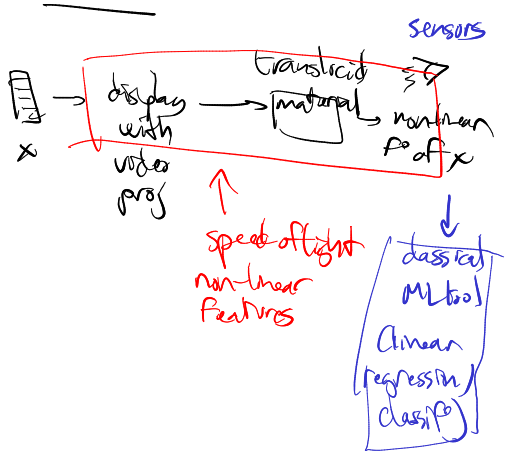
property: all the information is still there

Random weights: ≃ non-linear projection

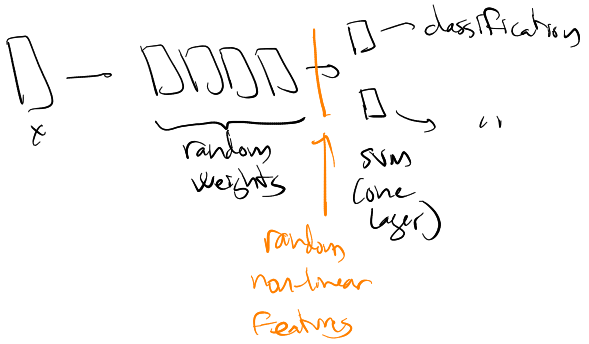


as many as neurons

Light-On



In some cases: most of the performance is due to the architecture and not to the training

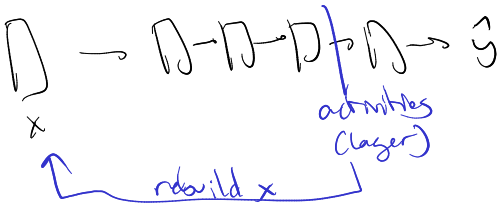


train everything
or
train only last layer
↓
same perf

(Extreme ML)

~~Why~~ work only for small architectures

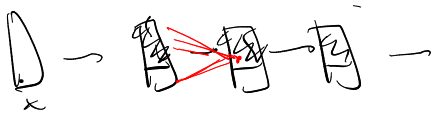
because the architecture was tuned beforehand



if random weights: good reconstruction of x → keep more information
if trained: kind of work (not as good) → keep only information relevant for the task

for deeper networks:

- in each layer: { 90% of neurons: randomly initialized (and fixed)
  { 10% of neurons: are trained



works

Architectural bias

- CNN: move to Fourier space

shown to learn low-frequencies first → looking at "the big picture" before at the details

- add layers $M^{l} M^{l-1} M \dots$ no gain in expressive power

but the distribution over functional space is different

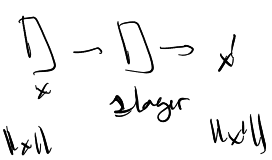
$M \sim \mathcal{L}$ $M_{ij} \sim \mathcal{N}(0,1)$ → $p(F)$

$M^{l} M$

Random initialization?

→ random: but which law?

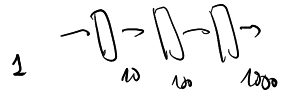
Exploding / vanishing gradients



activations' amplitude $\simeq$ input amplitude

$\|x'\| \simeq \gamma \|x\|$

$\gamma = 10$



#layers

factor γ^L

if 10 layers

20

100

$\Rightarrow$ issues

$\gamma = 0.1 \Rightarrow f \simeq 0$

$f \simeq \pm\infty$

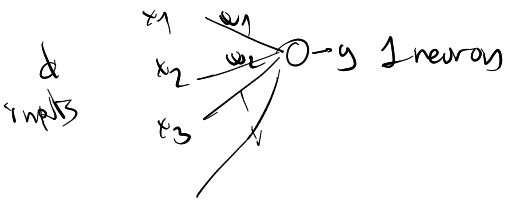
Same issue with gradients

$\nabla_\theta \simeq \pm\infty \Rightarrow$ not possible to train

$\nabla_\theta \simeq 0$

mitigate? Law s.t. $\gamma \simeq 1$

Xavier Glorot's initialization

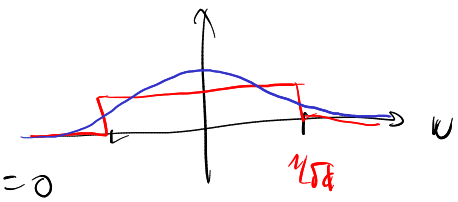


Hyp: inputs $x_i \sim \mathcal{N}(0,1)$ or just: mean = 0, variance = 1

$w_i \sim \mathcal{L}$

goal: find $\mathcal{L}$ s.t. variance(output) = 1

Solutions: $w \sim \mathcal{U}\left[\frac{-1}{\sqrt{d}}, \frac{+1}{\sqrt{d}}\right]$ or $w \sim \mathcal{N}(0, \sigma^2 = \frac{1}{d})$



Proof: $y = \sum_i w_i x_i$

mean: $\mathbb{E}_{x,w}[y] = \mathbb{E}_w\left[\sum_i w_i \mathbb{E}_x[x_i]\right] = 0$

variance: $\mathbb{E}_{x,w}[y^2] = \mathbb{E}_{x,w}\left[\sum_i w_i^2 x_i^2 + (\text{cross terms})\right] = \mathbb{E}_w\left[\sum_i w_i^2 \underbrace{\mathbb{E}_x[x_i^2]}_{1}\right]$

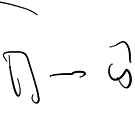
$= d\, \mathbb{E}_w[w^2]$

With activation function σ:

e.g.: ReLU → variance /2

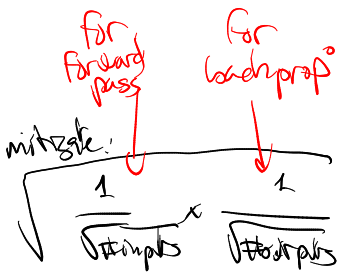
$\Rightarrow \times\sqrt{2}$

$\Rightarrow$ activation-specific corrector factor



Biases: initialized to 0

Other approaches: Kaiming He et al.



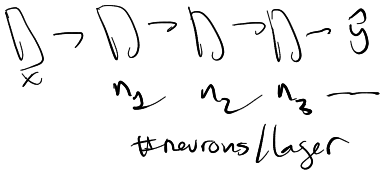
[Boris Hanin 2018] Jacobian properties

$J = \frac{df}{dx}$ well behaved at initialization

matrix dim outputs × dim inputs

$\frac{\partial y}{\partial x}$

regularizer $\left\| \frac{df}{dx} \right\|$



upon initialization

- $\mathrm{var}\left(\frac{df}{dx}\right) = 1/n_0$
- $\mathrm{var}\left(\left(\frac{df}{dx}\right)^2\right) = \mathbb{E}\left[\left(\frac{df}{dx}\right)^4\right]$

$\alpha e^{\sum_\ell 1/n_\ell} \leq \cdots \leq \beta e^{\sum_\ell 1/n_\ell}$

⇒ what matters: $\sum_\ell 1/n_\ell$ as small as possible

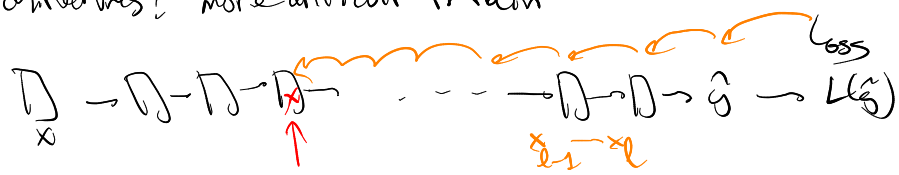
⇒ better → as many neurons as possible in each layer

↳ if on a budget: consider same-width layers (better than thin ones)

⇒ avoid thin layers (unless necessary, eg auto-encoder)

Design easy-to-train architectures

- deeper architectures: more difficult to train



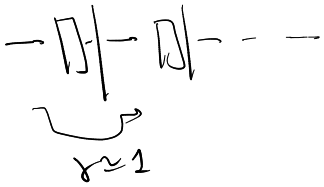
$\delta w \propto \frac{\partial L}{\partial \hat{y}} \frac{\partial x_\ell}{\partial x_{\ell-1}} \frac{\partial x_{\ell-1}}{\partial x_{\ell-2}} \cdots \frac{\partial x_1}{\partial w}$

$x_\ell = f(x_{\ell-1})$

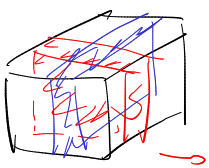
information $\frac{\partial L}{\partial \hat{y}}$ is lost

ok for few layers but not too many

- normalization
  - SELU activation fct: nice property: keeps the variance



- add scaling layers



compute (for each slice)
mean μ
s-d-i σ

normalize each slice of activities

Scaling parameter

$\gamma \frac{A_j - \mu}{\sigma} + b$

learned

batch-norm parameters

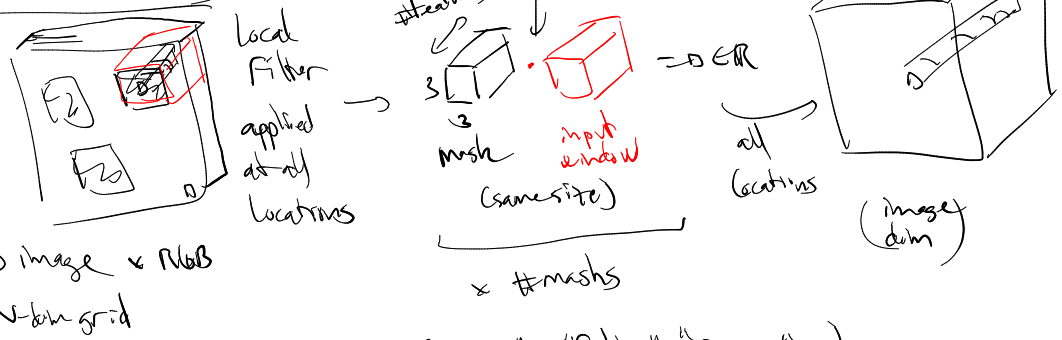
→ instance-norm

→ layer-norm

→ fix-up init

Architecture too

- CNN : convolutions

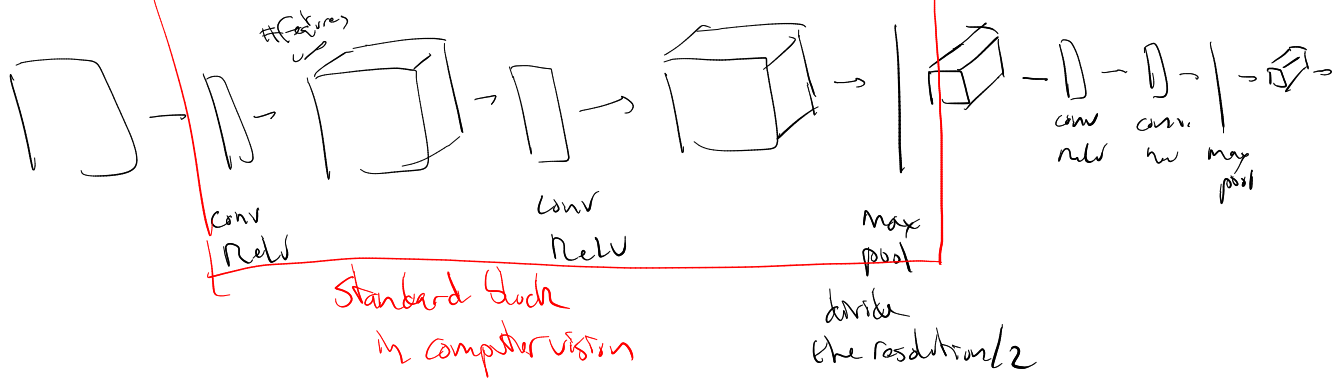


2D image x RGB

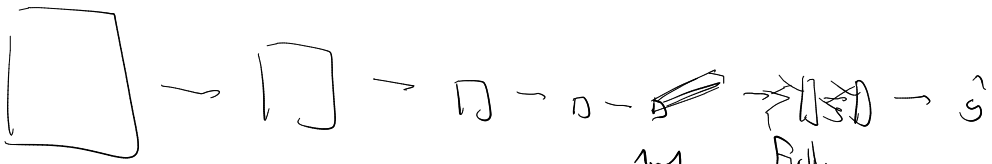
N-by-N grid

× #masks

("mask", "filter", "feature"...)



Standard block in computer vision



typical image classification pipeline

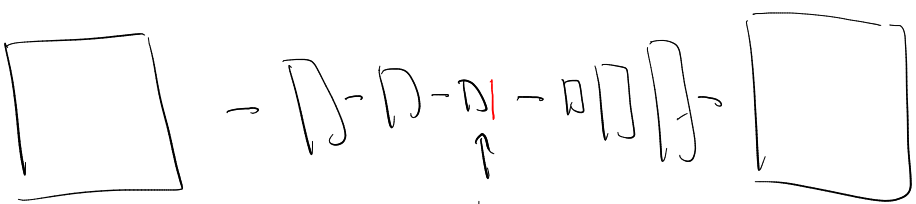
hierarchical model

more "semantic"

more details

Auto-encoders : unsupervised setup

"self-supervised" : build a supervised task from labels built from the input



x
input

smaller-dim representation
(dim reduction)
10 neurons

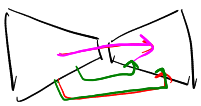
x'
output

$x' \simeq x$

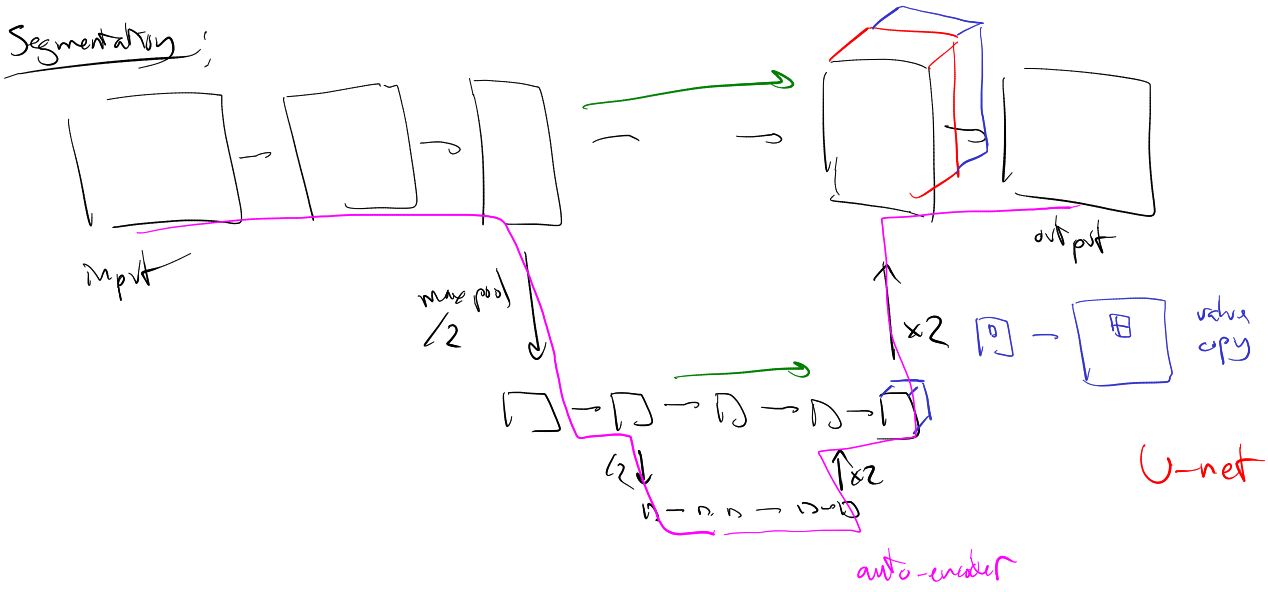
training criterion:

$\|x - x'\|$

generative models : cf chapter 6

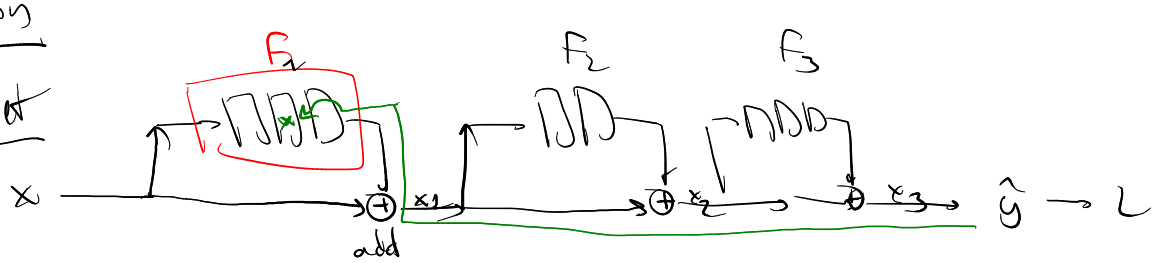


Segmentation:



Optimisation

ResNet



$x' = f(x)$

$x' = x + f(x)$

$\hat{y} = f_3(x_2) + f_2(x_1) + f_1(x) + x$

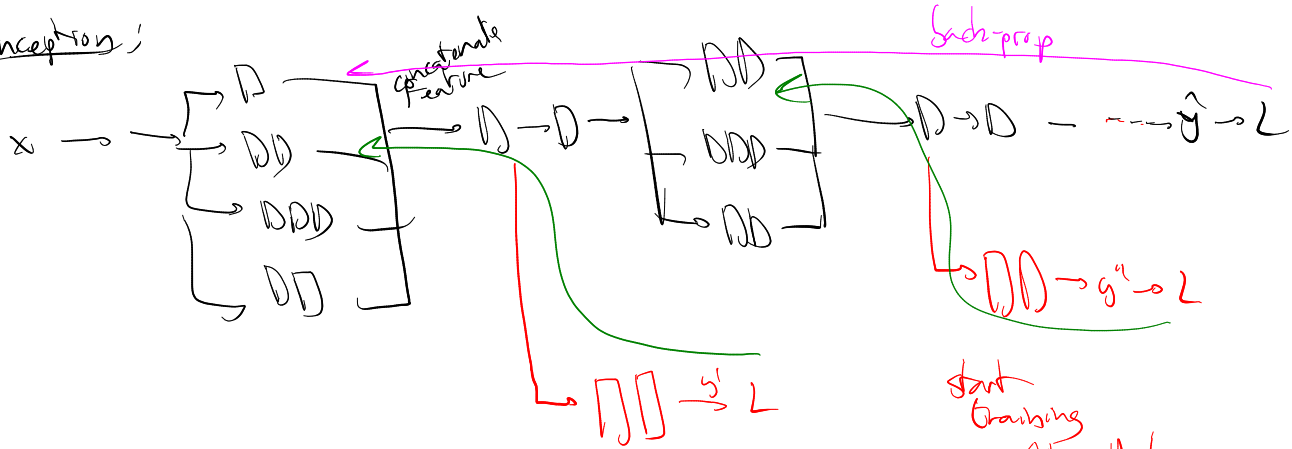
$$\frac{\partial L}{\partial w} = \frac{\partial}{\partial w}(\dots) + \frac{\partial L}{\partial \hat{y}} \frac{\partial f_1}{\partial w}$$

direct access to w !

$\Rightarrow$ much faster training

Auxiliary losses

Inception:

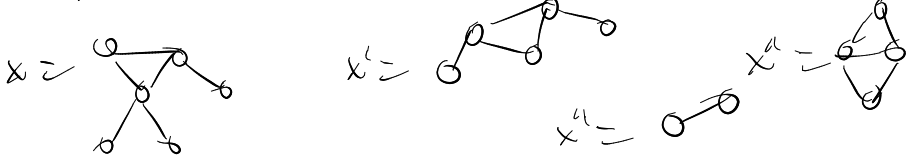


start training with all losses

then remove the red parts

Graph-NN

task: 1 input = a graph



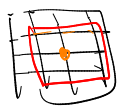
examples:

input: molecule

or: social graph

or: 3D simulation mesh

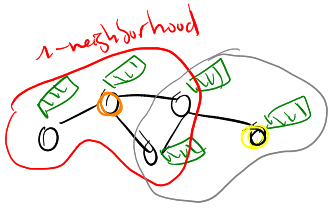
CNN

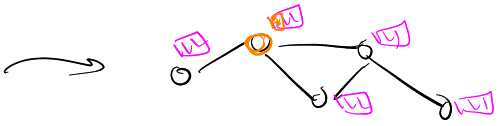


regular grid

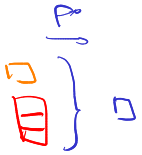
→ apply mask (filter)

or: surface mesh





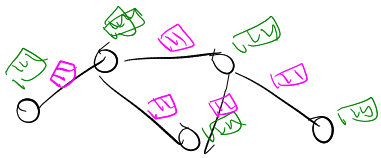
input x



functions; Flexible

output (of the layer)

$$y = w_{center} \cdot x_c + \frac{w_{neighs}}{\#neighbors} \sum_{n \in Neighbors} x_n$$



$$\begin{cases} y = w_c x_c + \sum_{n \in Neighbors} w(e_{n \to c})\, x_n \\ \text{also: compute new edge information} \\ e'_{a \to b} = f(x_a, x_b, \{e_{n \to a}\}, \{e_{n' \to b}\}) \end{cases}$$

edge-dependent

free design

very flexible

Spectral methods

Fourier domain

CNN 

$$(\square * x)(p) = \int_{p'} M(p')\, x(p - p')\, dp'$$

convolution

M

pixel location within the mask

Fourier

$\tilde{\square} \times \tilde{x}$

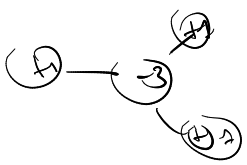
Fourier $x \to \tilde{x}$

multiplication of Fourier transforms

$\tilde{\square} \cdot \tilde{x}$

Fourier for graph?

↳ Laplacian Δ (graph)



Fourier

choice of mask = multiplicative weights for each frequency

↳ build the filters: eigenvectors of Δ → for surface matching graph

↳ " " : use $(\Delta \text{ matrix})^n$

makes sense only if edges are binary or distance-valued

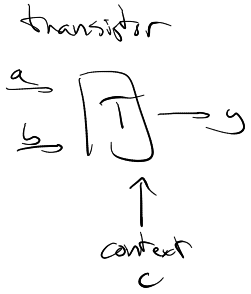
Δ : 

Δ^2 : 

Δ^n < n-neighborhood

Attention mechanism

- Basic :



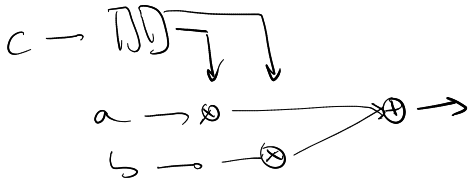
$y = w_a a + w_b b$

linear layer combination = fixed

if $c=0$: $1a + 0b$

if $c=1$: $0a + 1b$

$\Rightarrow w_a = f(c)$

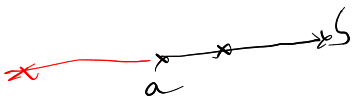


$w_a \geq 0$

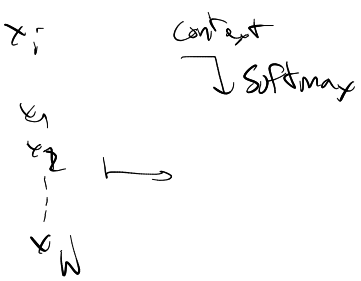
$w_b \geq 0$

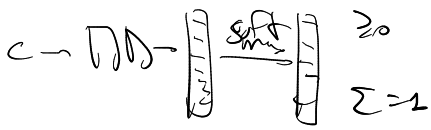
$w_a + w_b = 1$

interpolation of a & b

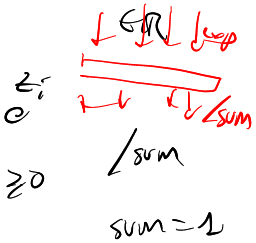


many inputs :

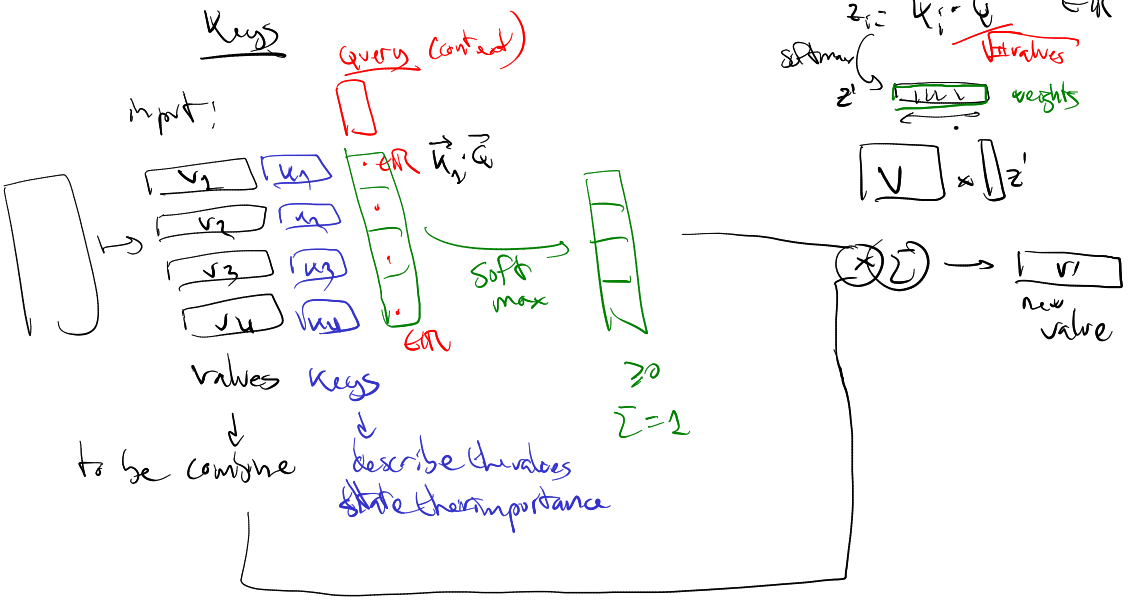




$\vec{z} = (z_1, \dots, z_N)$



Keys



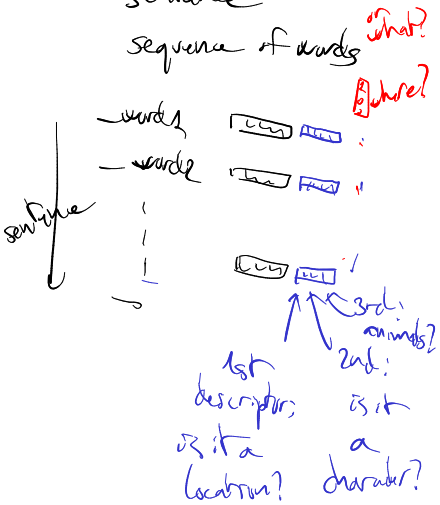
$z_i = \vec{k}_i \cdot \vec{q} \in \mathbb{R}$

$V \times z'$

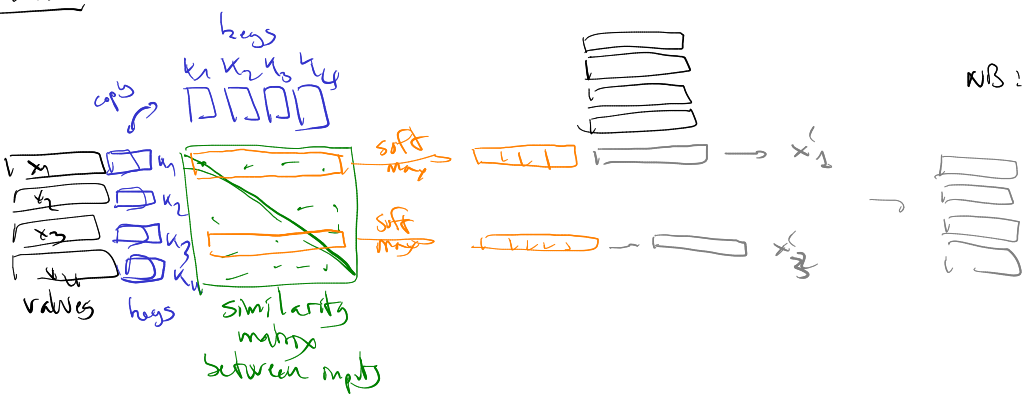
eg : NLP

sentence

sequence of words



Common use (transformers)



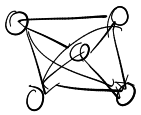
NB : can consider queries $q_1, q_2, \dots q_N$ different from keys $k_1, k_2, \dots k_N$ if relevant.

then :

$\text{softmax}\left(\frac{K \cdot q}{\sqrt{N}}\right) X$

≡ fully connected graph

each word = a node

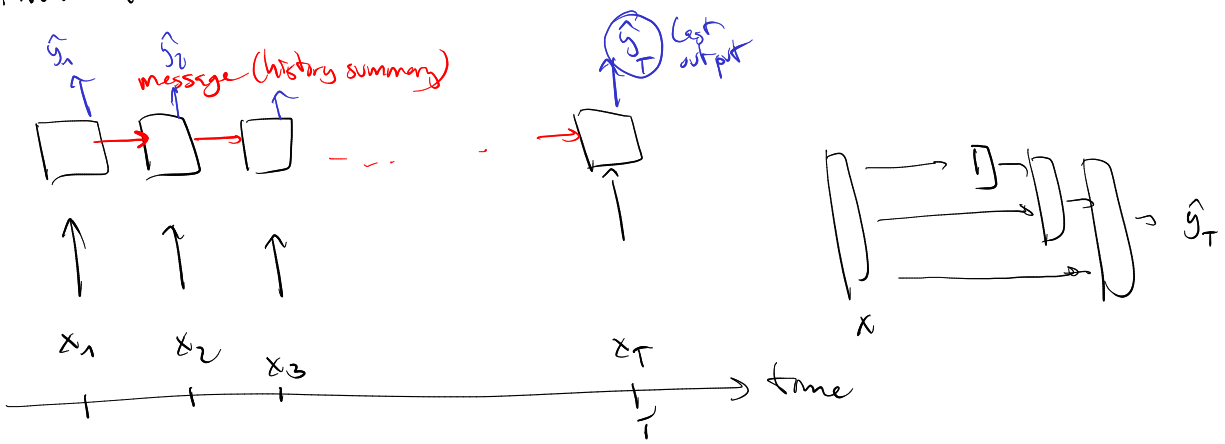


⇒ graph attention network (GAT)

⇒ listen to all nodes according to how similar they are to you (attention)

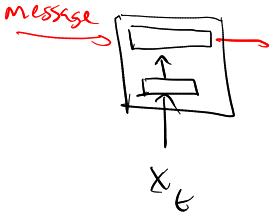
Recurrent networks

involve time : 1 input = time series $\vec{x} = (x_1, x_2 \dots x_T)$



all blocks are identical : share the same parameters

Block : based on attention GRU, LSTM



α message

$+(1-\alpha) f(x_t)$

$\alpha = g(\text{message}, x_t)$

"updating" the message

$\alpha \simeq 1$

or $\alpha \simeq 0$ if very important new event