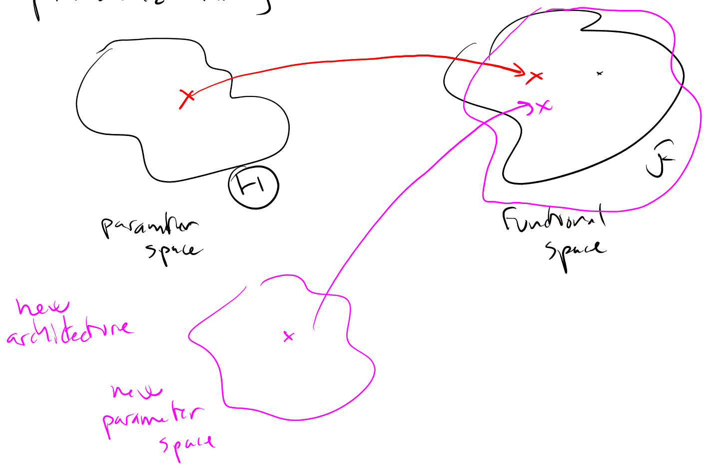
Architectures

I Architectures as priors on function space

Change of paradigm

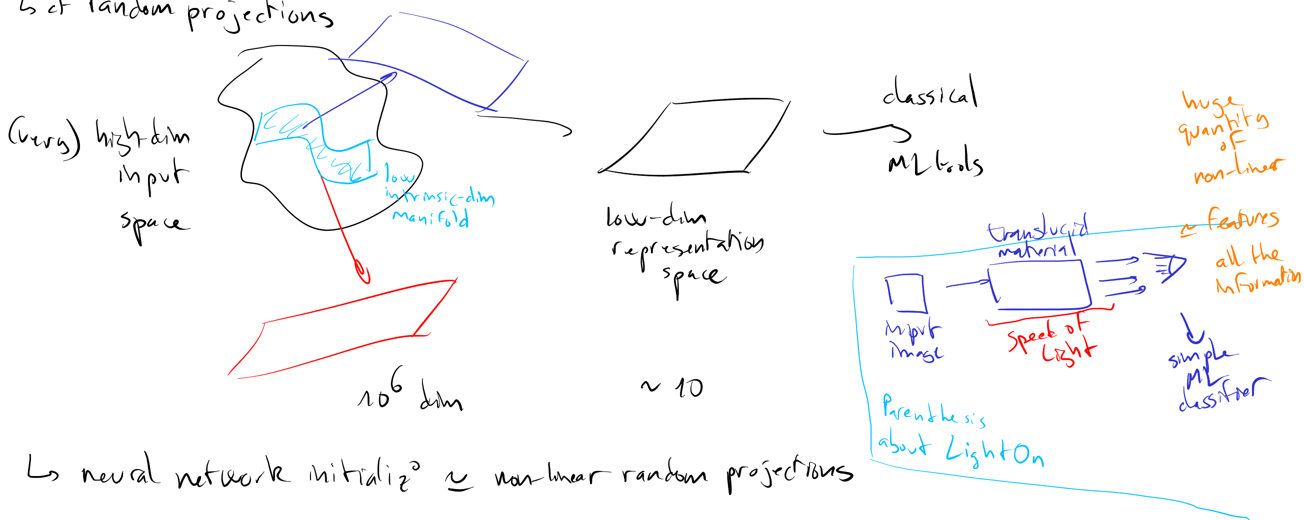
- classical ML: design features by hand
  vs deep L: meta-design of features → design architectures able to produce the features we think about

- an architecture = a parameterized family



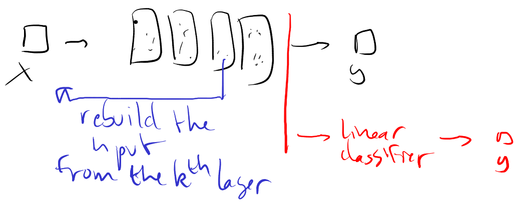
Architecture: prior on the function

- priori
  - as a constraint: what is expressible with this architecture?
  - most (reasonable) networks have already a huge expressive power
  - probabilities: easy to reach?
- with random weights: reasonable function or not?
  ↳ good architecture yields good features "naturally"
  ↳ cf random projections



↳ neural network initializ° ≃ non-linear random projections

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



5-layer LeNet

~~VGG~~

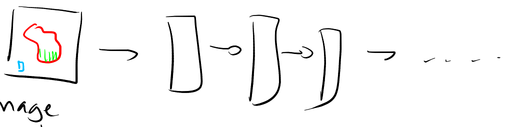
↳ Extreme Machine Learning: learn only the last layer ~~VGG~~

↳ quality of layers with random weights

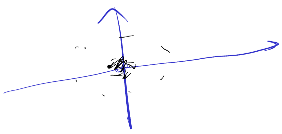
↳ quantify the information flow: information theory → "information bottleneck"

- Bias of the architecture

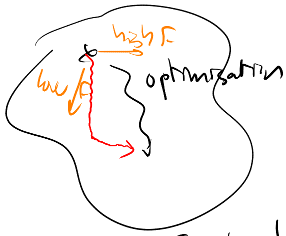
→ move to Fourier space (CNN)



Fourier

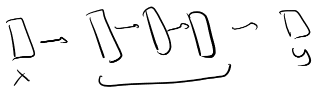


show that lower frequencies are learned first → Look at the "big picture" before the details



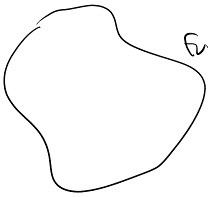
→ stacking fully-connected layers

Linear:



$$y = \underbrace{M_3 M_2 M_1}_{M'} x$$

expressive power = the same



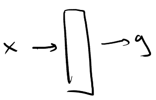
proba law: ex: $M_{ij} \sim \mathcal{N}(0,1)$

⇒ proba over M ↗ distribution of $M' \neq$ of M

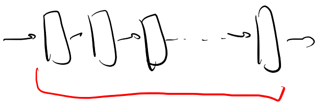
Random initialization:

→ random, according to which law?
↳ choose a law yielding good properties

* exploding/vanishing gradient



$\|y\| \simeq \gamma \|x\|$ $\gamma \in \mathbb{R}^+$ magnifying factor



$\|y\| = \gamma^L \|x\|$
↳ $\gamma < 1$: $y \simeq 0$ Function $\simeq 0$
↳ $\gamma > 1$: $y \simeq +\infty$
⇒ wish for good start

Same for backpropagation

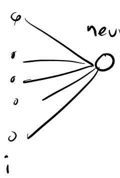
$$\|\nabla_a F\| \simeq \underbrace{(\gamma')^L}_{\text{prevent optimiz}^\circ} \left\|\frac{dL}{dF}\right\|$$

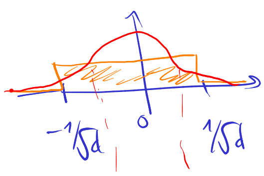
⇒ wish for good optim° properties

· Xavier Glorot initialisation

· $w \sim \mathcal{U}\left[-\frac{1}{\sqrt{d}}, \frac{+1}{\sqrt{d}}\right]$

or $\sim \mathcal{N}\left(0, \sigma^2 = \frac{1}{d}\right)$



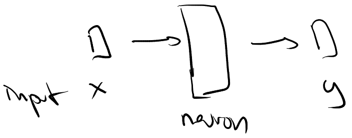


· justification: $\begin{cases} w_i \text{ iid of mean 0, standard deviation } 1/\sqrt{d} \\ x_i \text{ iid of mean 0, variance 1} \end{cases}$

neuron computes: $y = \sum_i w_i x_i$ → statistical properties?

↳ mean: $\mathbb{E}_x \mathbb{E}_w [y] = \mathbb{E}_w \left[\sum_i w_i \overset{0}{\mathbb{E}_x[x_i]}\right] = 0$

↳ variance: $\mathbb{E}_{x,w}[y^2] = \mathbb{E}_x \mathbb{E}_w \sum_i w_i^2 x_i^2 = \underset{\text{over } d \text{ inputs}}{\sum_i} \underbrace{\mathbb{E}_{x_i}[x_i^2]}_{1} \underbrace{\mathbb{E}_w[w_i^2]}_{1/d} = 1$

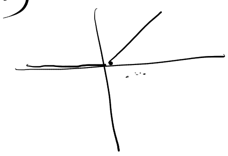


$x_i \sim$ law mean 0, σ 1 . . . . $y \sim$ law mean 0, σ 1

$\Rightarrow \gamma = 1 \rightarrow$ no exploding/vanishing issues (reasonable outputs)

+ non-linear activation: $y = \sigma\left(\sum w_i x_i + b\right)$

↳ ex: ReLU



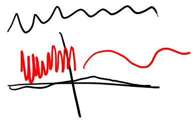
→ divides variance by 2
⇓
correct with a factor $\sqrt{2}$

- bias b? → initialize to 0

. other initialization laws (similar properties): He et al

[Boris Hanin, 2018] Jacobian properties

→ at initialization, $\underbrace{\dfrac{dF}{dx}}$ Jacobian matrix: dim outputs × dim inputs



$\left\| \dfrac{dF}{dx} \right\|$

- n_ℓ : number of neurons in layer ℓ
- $\mathrm{var}\left(\dfrac{dF}{dx}\right)$ is fixed $= 1/n_0$
- $\mathrm{var}\left(\left(\dfrac{dF}{dx}\right)^2\right) = \mathbb{E}\left[\left(\dfrac{dF}{dx}\right)^4\right]$

$\beta e^{\sum_\ell 1/n_\ell} \leq " \leq \alpha\, e^{\underline{\sum_\ell 1/n_\ell}}$

$x \rightarrow [\,]_{n_1} \rightarrow [\,]_{n_2} \rightarrow [\,]_{n_3} \rightarrow y$

⇒ better to choose equal widths than one thin layer

⇒ put sufficiently-many neurons in each layer

Designing architectures easy to train

→ "deep" networks: not that deep in terms of optim°

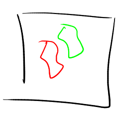
$x \rightarrow [\,][\,][\,][\,] \rightarrow y \rightarrow \text{Loss}$

often: small number of layers to cross ⇒ next part

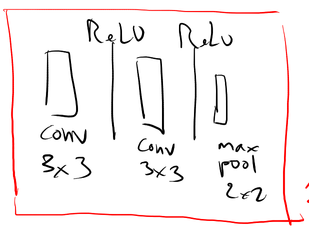
→ need "depth"?

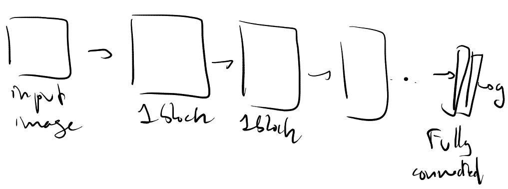
II Architecture zoo

For vision problems: CNN → share local filters ⇒ invariance to translation
↳ hierarchical model ⇒ much greater generaliz° power



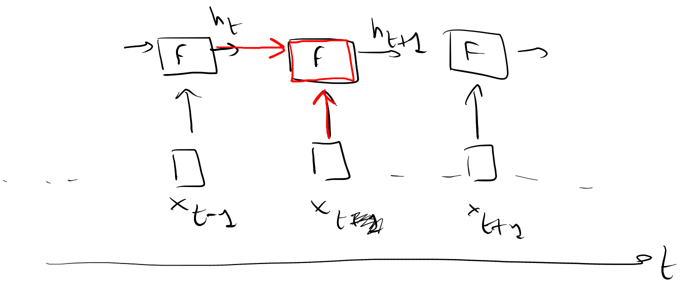
typical block:





Recurrent networks:

- Basic RNN: $h_{t+1} = f(h_t, x_t)$

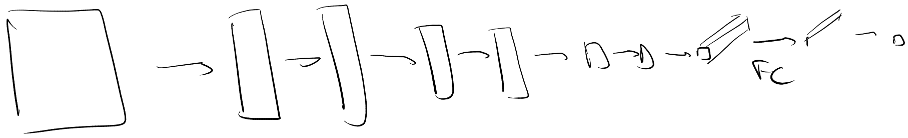


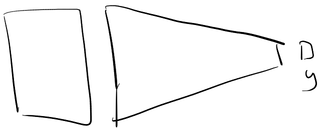
- memory
- leaky: $h_{t+1} = h_t + f(h_t, x_t)$
- gates

⇒ LSTM
GRU
MinimalRNN

Scale & resolution

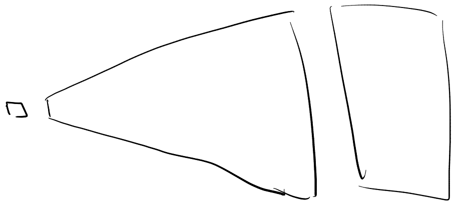
- classification tasks in computer vision: pyramidal approach



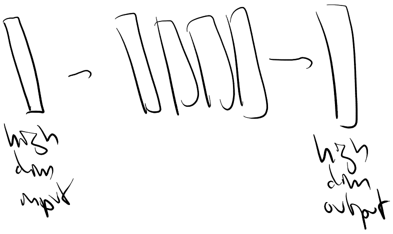


VGG AlexNet LeNet

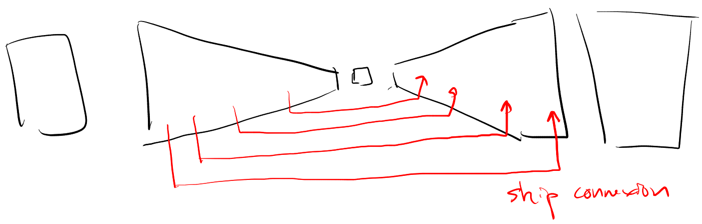
- generation:



- regression / high input/output dim

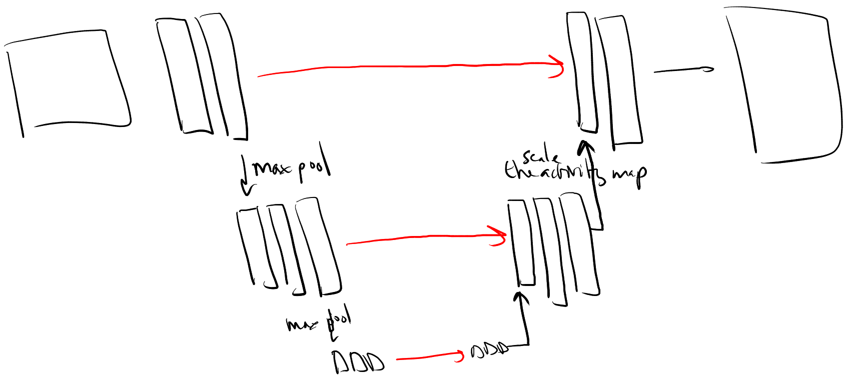


ex: transformed image (of same resolution)



lose the details

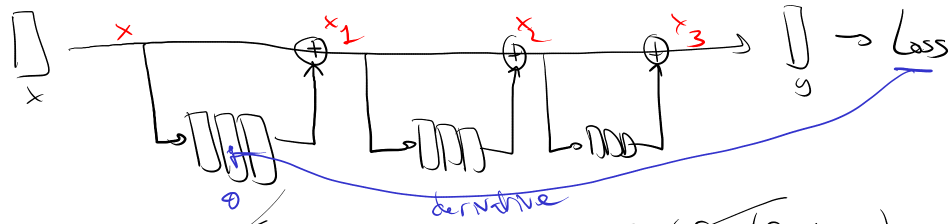
U-net



ex: segmentation

Depth & mixing blocks

ResNet:



$\cancel{x' = f(x)}$

$x = x + f(x)$

$\cancel{y = f_3(f_2(f_1(x)))}$

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

$$\frac{\partial L(y)}{\partial \theta} = \cancel{\frac{\partial L}{\partial f_3} \times \frac{\partial f_3}{\partial f_2} \times \frac{\partial f_2}{\partial f_1} \times \frac{\partial f_1}{\partial \theta}}$$

$$= \frac{\partial L}{\partial y} \times \left(\underbrace{\frac{\partial f_1}{\partial \theta}}_{\text{direct feedback}} + \frac{\partial f_2}{\partial f_1}\frac{\partial f_1}{\partial \theta} + \frac{\partial f_3}{\partial f_2}\left(\frac{\partial f_2}{\partial f_1}\frac{\partial f_1}{\partial \theta} + \frac{\partial f_1}{\partial \theta}\right) \right)$$

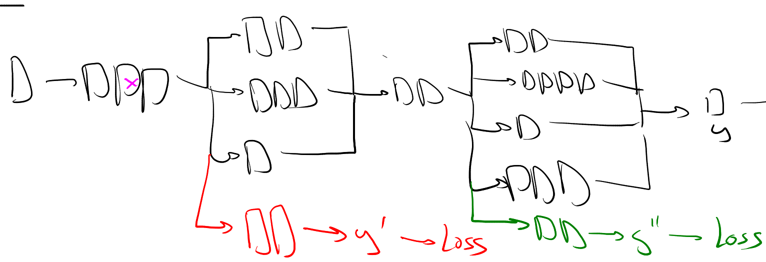
Highway networks

→ same idea + attention

Orthogonal matrices

→ train a feedforward network: 10^4 layers

Inception



Auxiliary losses

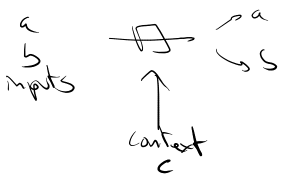
Start training: Loss(y) + Loss(y') + Loss(y'')

↓

Loss(y) / / when optimizer goes fine

Attention

· basic ex: transistor:



$y = w_a a + w_b b$

Linear combination

Above: Fixed

$$y = \underbrace{w_a(c)}_{f_1(c)} a + \underbrace{w_b(c)}_{f_2(c)} b$$

α $(1-\alpha)$ interpolation

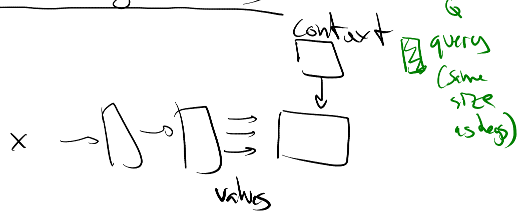
$y = \alpha a + (1-\alpha) b$

$\alpha = f(c)$ $\alpha \in [0,1]$

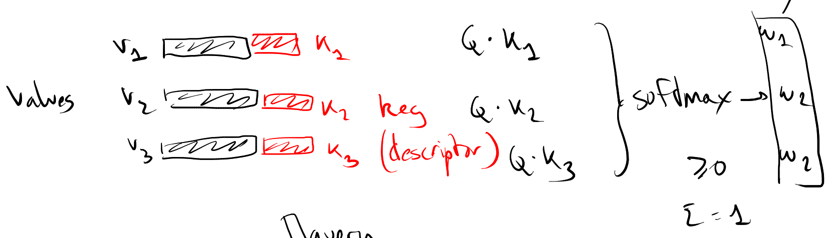
$= \text{softmax}(\cdot c)$

sigmoid

Attention (general case)



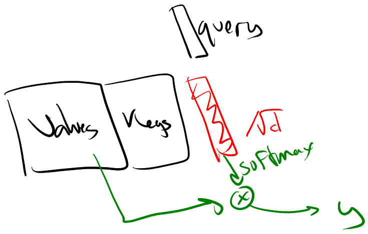
$\sum w_i v_i$ — w_i dep(c) → $Q \cdot K_i$; $v_i \in \mathbb{R}^d$



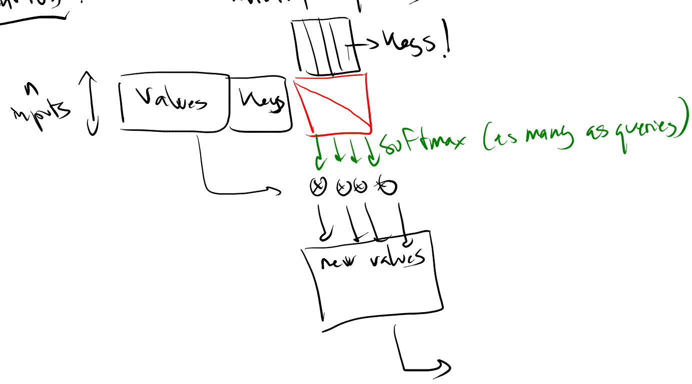
$$y = \sum_i^d v_i \times s\left(\frac{Q \cdot K_i}{\sqrt{d}}\right)$$

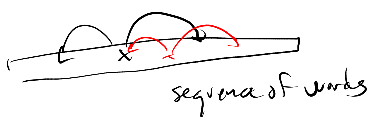
softmax

to avoid saturation of the softmax



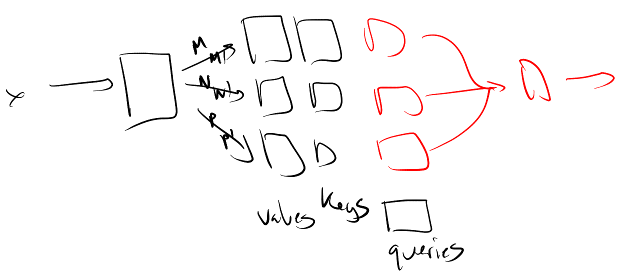
Self-attention:





Multihead:

multiple attention blocks in parallel



multihead

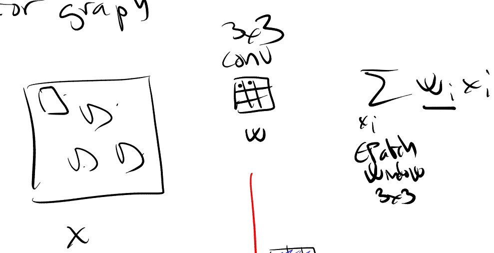
$V = Mx$ $\quad Nx \quad Px$

$K = M'x$ $\quad N'x \quad P'x$

Q { Fixed (parameters)
$M''x$ or P''(other things)

Graph-NN (GNN)

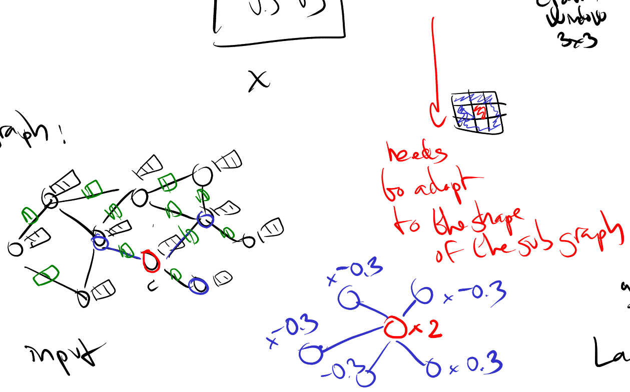
- analogy of conv for graph
- computer vision



$\sum w_i x_i$

x_i patch window 3×3

- any graph!

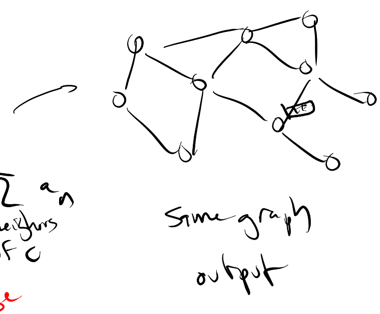


parameters

$y_c = 2a_c + (-0.3) \sum_{\text{neighbors of } c} a_n$

or average

Layer



for each node: $F(a_c, \{a_{neighbors}\})$

↳ Function needs to be generic to adapt to varying # of neighbors

using edge information:

$$F(a_c, \{(a_i, e_{i \to c})\}_{i \in neighbors(c)})$$

$$y_c = 2a_c + \underbrace{\sum_{i \in N(c)} w_i(e_{i \to c})\, a_i}$$

↳ towards attention mechanism

$M \, e_{i \to c}$

↑ parameters

↳ GAT: graph attention network

⇒ very flexible architecture

$\left\{ \begin{array}{l} \text{set of nodes with activities} \\ \text{set of edges with activities} \end{array} \right. \xrightarrow{F} \left\{ \begin{array}{l} \text{new values for nodes} \\ \text{new activities for edges} \end{array} \right.$

same graph

→ keeps the same graph

↳ graph classification?

input: graph $\xrightarrow{}$ output: $\in \mathbb{R}$?

decrease regularly the size of the graph

or FC