Texas A&M Institute of Data Science Tutorial Workshop Series

Introduction to Deep Learning

by Boris Hanin

June 12, 2020

Deep Learning Tutorial

Goal: ① What is a NN?

② How NNs are used?

③ What kinds of choice are made by engineers?

④ Main use cases and failure modes?

Supervised Learning

① Data Acquision: obtain dataset

$\mathcal{D} = \{(x_k, f(x_k))\}_{k=1}^{K}$ (f unknown)

- e.g. x_k = image ; $f(x_k) = \begin{cases} 1, & x_k \text{ has STOP} \\ 0, & \text{o/w} \end{cases}$

② Model Selection: Choose model

$$x \longmapsto \mathcal{M}(x; \Theta)$$

where Θ = vector of params

- e.g. $\mathcal{M}(x;\Theta) = a_1 x_1 + \ldots + a_n x_n$, $\Theta = (a_i)$

③ Optimization: Use $\mathcal{D}$ to obtain setting Θ_* s.t.:

$$f(x_k) \approx \mathcal{M}(x_k; \Theta_*)$$

- e.g. linear regression

④ Testing: See how well Θ_* performs on unseen data:

$$¿\ f(x) \approx \mathcal{M}(x; \Theta_*)\ ?$$

Salient Features:

- input dim $\gg 1$ $(10^2 - 10^6)$
- #params $\gg$ #data $\gg 1$
- NNs both interpolate and extrapolate

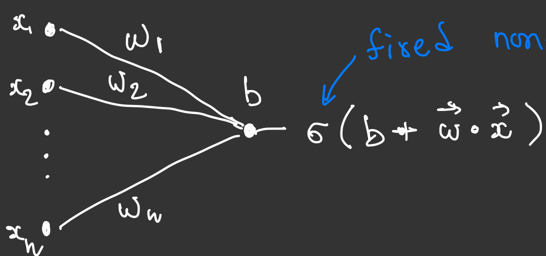
Neural Nets:

- Neural nets are built of "neurons"

$$x=(x_1,\dots,x_n) \longmapsto z(x;\theta) = \sigma(b + x_1w_1+\dots+x_nw_n)$$

$$\theta = (\underset{\text{bias}}{b}, \underbrace{w_1,\dots,w_n}_{\text{weights}})$$

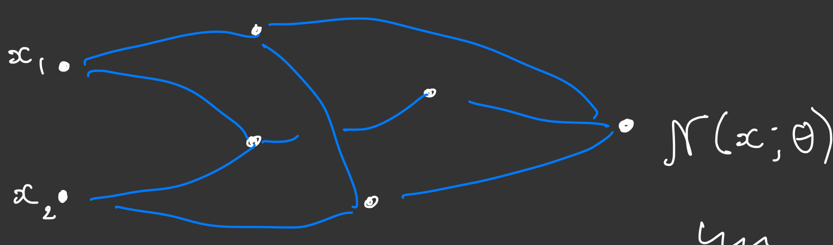
fixed non-linear function



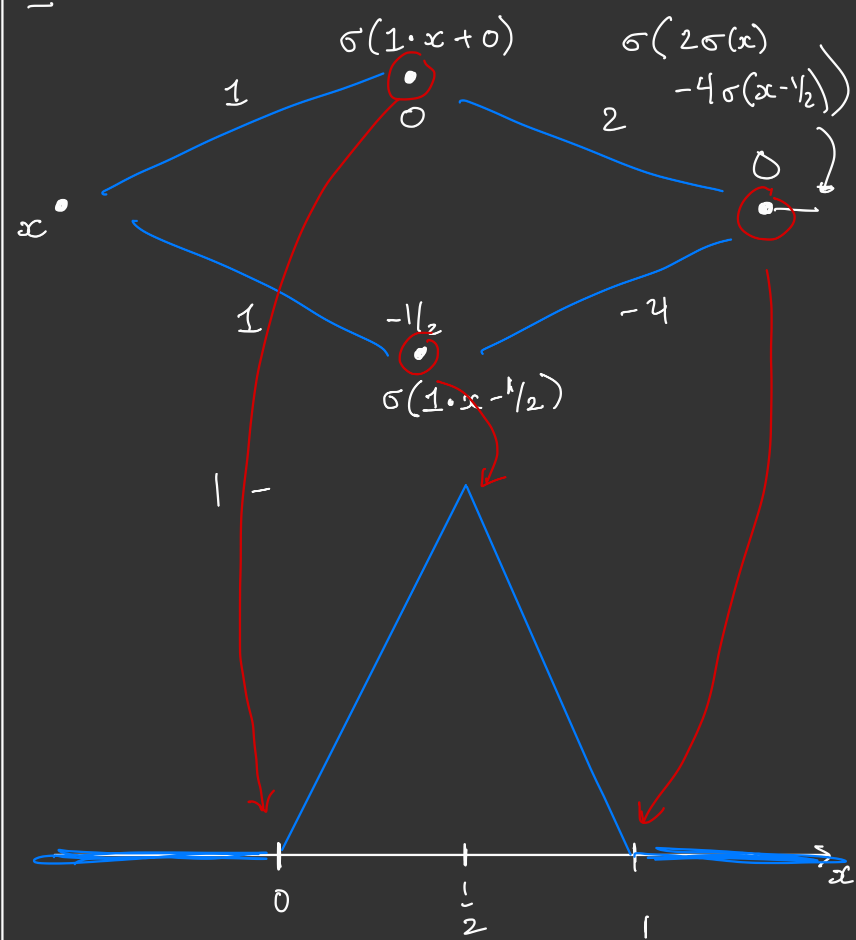
"integrate & fire"

- "Def" A neural network is a collection of neurons and wiring diagram

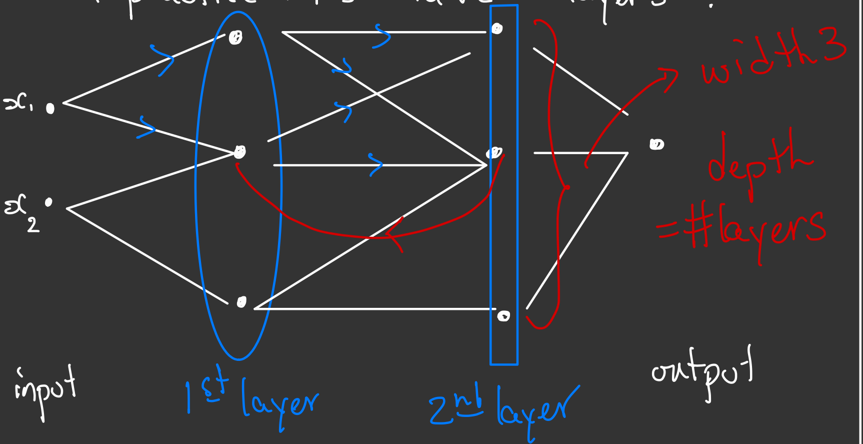
"archtiecture" $\qquad \theta = (\text{all } b\text{'s}, W\text{'s})$



Ex: $\sigma(t) = \mathrm{ReLU}(t) = \max\{0,t\}$



- In practice NNs have "layers":



$$x \mapsto x^{(1)} \longmapsto x^{(2)} \mapsto \mathcal{N}(x;\Theta)$$

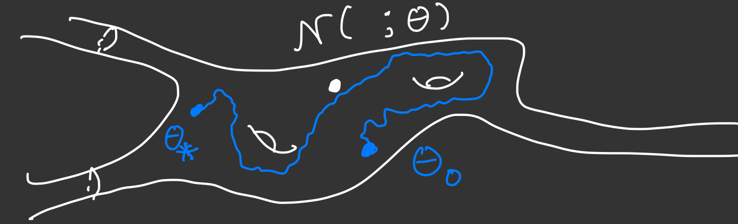
- layers $\longleftrightarrow$ heirarchical reps

<u>Typical Use:</u>

① <u>Data Acquisition</u>: $\mathcal{D} = \{(x_k, f(x_k))\}$, $x_k \in \mathbb{R}^{n_0}$, $n_0 \gg 1$

② <u>Architecture Selection</u>: choose σ's, wiring diagram, depth, width

③ Randomly initialize $\Theta = \{W, b\}$



③' Optimize Θ by gradient descent on

$$\mathcal{L}_{\mathcal{D}}(\Theta) = \frac{1}{|\mathcal{D}|}\sum_{x \in \mathcal{D}} \ell_x(\Theta) \ ; \quad \ell_x(\Theta) = |f(x) - \mathcal{N}(x;\Theta)|^2$$

④ <u>Testing</u>: Draw new $(x, f(x))$ and check whether

$$\boxed{\mathcal{N}(x;\Theta_*) \approx f(x)}$$

<u>Empirically</u>: deeper is better*

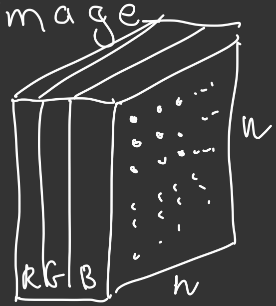
Main Uses:

① NLP

- x_k = "the cat is big" *
- $f(x_k)$ = "le chat est grand"
- Google Translate
- Siri
- Chat Bots

② Computer Vision

- x_k = image



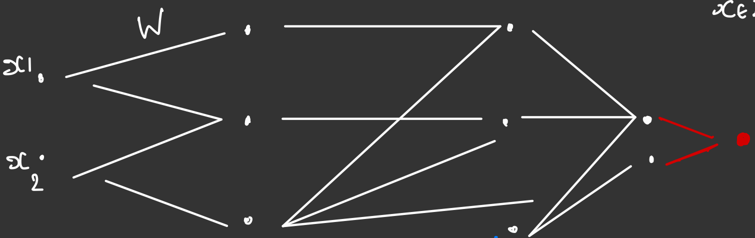
- $f(x_k) = \begin{cases} 1, & x_k \text{ has human} \\ 0, & \text{o/w} \end{cases}$
- Self-Driving Cars
- Facial Rec

③ Reinforcement Learning

- x_k = state of system
  (e.g. position of chess board)
- $f(x_k)$ = reward-max action
  (e.g. best next move)
- AlphaGo
- Exploration by AU

Neural Net Optimization

$\mathcal{L}_D(\theta) = \frac{1}{|D|} \sum_{x \in D} \ell_x(\theta)$



$N(x;\theta)$

- $\underline{GD}$: $\Delta W = -\lambda_W \frac{\partial \mathcal{L}_D}{\partial W}$ (learning rate: λ)

- Compute $\frac{\partial \mathcal{L}_D}{\partial W}$ using "backpropagation" (i.e. chain rule)

- $\underline{SGD}$:
  - randomly select "batch" $B \subseteq D$
  - replace $\mathcal{L}_D$ by

$$\mathcal{L}_B = \frac{1}{|B|} \sum_{x \in B} \ell_x(\theta) \approx \mathcal{L}_D$$

batch size: $|B|$

- $\underline{good}$:
  - small batches mean less computation

- Key: How to choose λ, $|B|$?

Intuition:

① small $\lambda \longleftrightarrow$ slow but accurate
large $\lambda \longleftrightarrow$ fast but noisy

② Why choose same λ for all params? $N(x;\theta)$ might be very sensitive to θ_1 but not to θ_2.

③ In practice: find λ by grid search on log-space

④ Why keep λ constant during train?

Schedules:
- $\lambda = \text{const}$
- λ decays ($\lambda \leftarrow .9\lambda$) every fixed # steps
- λ - periodic $\lambda(t) = |\cos(2\pi k t)|$

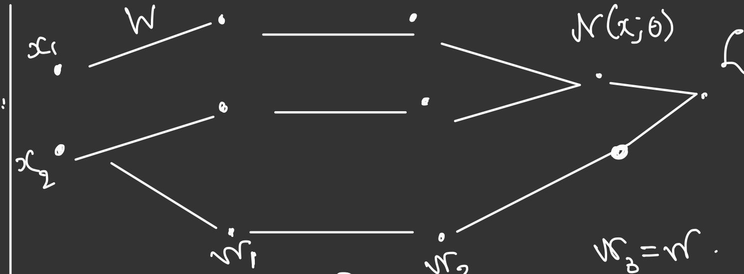
⑤ small $|B| \longleftrightarrow$ noisy but fast
large $|B| \longleftrightarrow$ accurate but slow

$[\lambda \cdot |B|^{1/2} \equiv \text{const}]$

⑥ λ, $|B|$ are inversely related

Architecture Selection:

- Best architecture is always data-dependent:
  - NLP $\longleftrightarrow$ $\begin{cases} \text{Tranformer-based} \\ \text{Recurrent (LSTM + Attention)} \end{cases}$
  - CV $\longleftrightarrow$ Convolutional, Residual
- Still leaves many choices:
  - details of wiring (width/depth...)
  - choice of σ (= ReLU)
  - how to initialize and to optimize
- Empirical: deep is good*



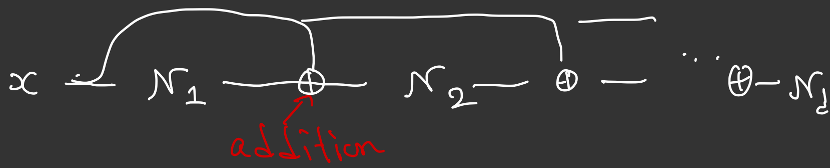
$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial v} \underbrace{\frac{\partial v}{\partial v_2} \frac{\partial v_2}{\partial v_1}}_{\text{product } \approx \#\text{layers Jacobians}} \frac{\partial v_1}{\partial W}$$

- $d \gg 1$: $\left| \frac{\partial \mathcal{L}}{\partial W} \right| \simeq 0$ or $+\infty$

  "exploding/vanishing gradients"

$*$ = but often less stable

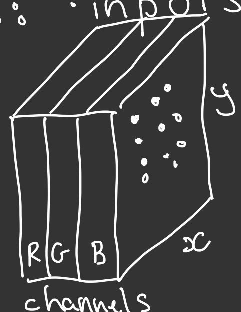
Residual Network:



$$\text{output} = x + N_1(x) + N_2(x + N_1(x)) + \dots$$

Intuition: N_j = j^{th} order correction to $x \mapsto x$

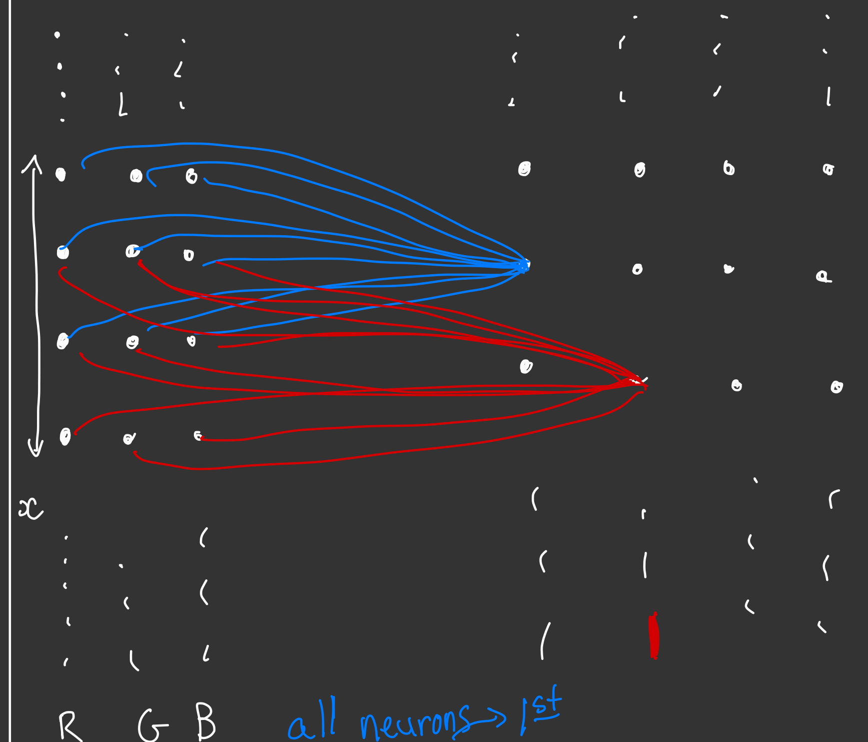
ConvNets: · inputs are nxn RGB



Key: Images are heirarchical and local

· In a ConvNet every layer has channels x (x, y) structure

· Every neuron in layer shares weights



all neurons → 1st look for same pattern

Challenges

① New Use Cases:

- PDE (fluids, physics, chemical...)
- Biology (genomics)

② Distribution Shift (nature of data change):

- change in hardware
- sunny vs. cloudy
- issue: NNs tend to be brittle

99.9% ⟶ 99.9% hydrant