Deep Learning in Practice

Chapter 1 : Deep Learning vs. Classical ML & optimisation

I Going Deep or not?

- no guarantee
- Universal approximation theorems
  - [Cybenko, 1989] [Hornik, 1991] :
  - [Sprecher, 1965]
  - [Kolmogorov, 1956] : exact decomposition of f by tuning the activation function

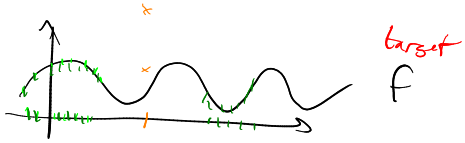
target f^* : $f : [0,1]^d \to \mathbb{R}^{d'}$ — precision $\forall \varepsilon$, number of neurons $\exists N$,

neural net : g_θ — input $\in \mathbb{R}^d$ — σ

$\|g_\theta - f\| \leq \varepsilon$

$$\forall f \in C^0, \ \exists \sigma, \ N < +\infty, \ g_{\theta,\sigma,N} = f$$

Being able to approximate ($\exists \theta, \dots$) $\neq$ being able to learn & generalize



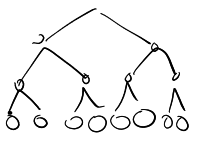
- Depth simplifies the approximation/estimation task
  - [Why does deep & cheap learning work so well?]

ex : n input variables

input $x \in \mathbb{R}^n$ target : $\prod x_i$

→ provided σ is smooth, possible to approximate multiplication of 2 variables with 4 neurons

→ for n variables :



binary tree, $\log n$ depth, $4n$ neurons

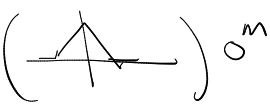
a, b

$\sigma(a+\varepsilon) = \sigma(a) + \varepsilon\sigma'(a) + \frac{\varepsilon^2}{2} \dots$

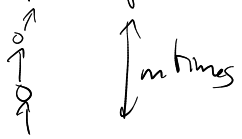
↳ without only 1 hidden layer : 2^n neurons at least. → solves only the case of binary inputs $(0, 1)$

- [Telgarsky]

ex : target function :

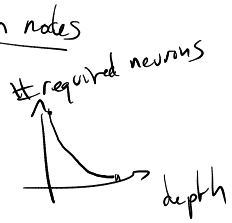
$\equiv$  $\circ^m$ 

$\Rightarrow$ representable with a thin deep network (depth : m, 1 neuron)

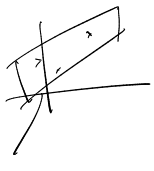


↳ with a Flat network (2 layers) : need 2^{mC} nodes

↳ $\sqrt{m}$ layers → $2^{\sqrt{m}}$ nodes



- [Poggio 2016]

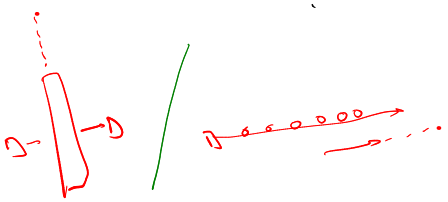


$f(\vec{x}) = \vec{w} \cdot \vec{x} + b$

d parameters to tune → $\propto d$ samples

$f(\vec{x})$ = function of $\vec{x}$ & of d parameters

↳ need $\propto d$ samples



If model not known : $\exp(d)$ samples for a Flat network

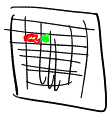
- Deep is sufficient (without width) [Lin & Jegelka, 2018] . ResNet with 1-neuron layers → universal approximation property

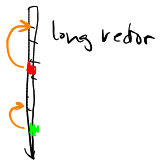
- Does it work? When?

task: image classification

→ big success: computer vision & NLP

↳ convolution: explicit geometry handling
translation invariant
hierarchical models





≠ random forest: can't incorporate geometry (yet)

- success in Reinforcement Learning

Alpha-Go Alpha-0

- 4.6 million training games ≫ one life ~ humanity
- 4 hours training on ... 5000 TPU > 2 years of 1 TPU ~ 1000s years on single CPU
↓
quite slow training

↳ computational training cost
NLP: huge networks (BERT, XLnet ...) → ≃ 100 000 $

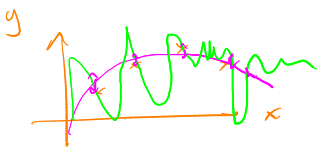
2) Gap between deep Learning & classical ML/optimization

- Reminder: ML setup
  - dataset: (x_i, y_i)
  - estimate f: $\forall x_i \mapsto y_i$
  - loss function Loss: quantifies the quality of the candidate f

↳ $$\inf_{f \in \text{predefined parameterized family}} \sum_{\text{examples } i} \text{Loss}(f(x_i), y_i) + \text{Regularizer}(f)$$

$\|\frac{df}{dx}\|^2$

↳ make f smooth ⇒ better generalization properties



↳ Minimum Description Length principle (MDL)
↳ Occam's razor

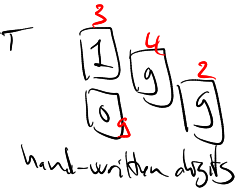
- Now with deep Learning:
  - millions of parameters
  - possible to train networks with 10^6 parameters without overfitting → still, good generalize° gap between train & test errors
  - and this with fewer samples than parameters
    ↳ estimator convergence?
  - classical optimizer°: the fewer parameters, the better
    ↳ highly non-convex optim°: thought to be very hard
  - add noise in the optim° process ⇒ helps!
  - training criterion: cross-entropy
    but evaluate on another one: accuracy ← not differentiable
  - common recommendation: check that the model is able to overfit
  - no regularizer!

* Closer look at overfitting

- huge models don't necessarily overfit

- [Understanding DL requires rethinking generalization, 2017]
  - Theorem: n samples $\in \mathbb{R}^d$
    $\exists$ 2-layer neural network, with ReLu activations and $2n+d$ weights that can associate each sample with a randomly-chosen target

Ex: MNIST



random labels

100% accuracy for these random labels

1000 samples
$d \simeq 1000$

even very small networks are completely able to overfit

$\Rightarrow$ capacity is not the issue
(Rademacher, Vapnik-Chervonenkis)

- how to detect overfitting? / how to prevent it?
  - possible regularizers: weight decay, drop out, data augmentation, ... } not clear: not sufficient, not necessary
  - easy stopping
  - SGD (optimizer) regularizes

Palliatives for regularization

- What about functional regularizers? i.e. $\int \left\| \frac{df}{dx} \right\|^2 dx$

[Bruschhäuser-Stamm]: first step: check that the norm is 0?
i.e. $f = 0$ everywhere $[0,1]^d$
NP: all inputs are booleans

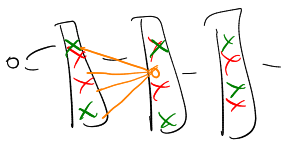
$F: \{(0,1,1,0\ldots)\}$

$\{0,1\}^d \longrightarrow \mathbb{R}$ $\quad f \overset{?}{=} 0$

↳ NP-hard

↳ stochastic approximation of the norm: Feasible

- Drop out training step t: pick randomly half of the neurons $\to 0$



→ improve the performance (↘ overfit)

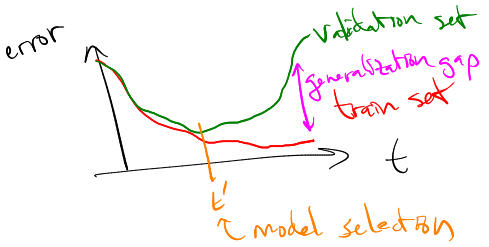
half of inputs (of that neuron) are missing $\Rightarrow$ robustness to inputs → redundancy

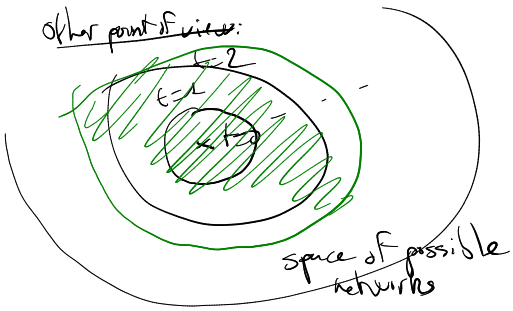
activity a_i (neuron i) $\quad \frac{df}{da_i} \simeq 0$ ⇝ not far from $\left\| \frac{df}{dx} \right\|$

- [Dropout as a Bayesian Approximation]
  ↳ each drop out realization: a $\neq$ sub-network
  ↳ Family of (sub-)networks
  ≥ ensemble technique

Early stopping

dataset → training, validation, test] only once!





Optimiz° noise

[Possio]

- around convergence: Hessian: $\frac{d^2 f}{dx^2}$

↳ if $H > 0$ (definite positive) [all eigenvalues > 0] : SGD noise will not harm

↳ if not: Loss = $\sum_{\text{training set}}$ errors $\Rightarrow$ error $\simeq$ stable for points in training set

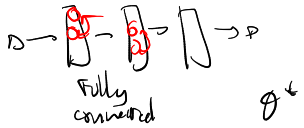
↳ error ↗ for test points

$\Rightarrow$ SGD overfits after a while

- be sure that $H > 0$:
  add a weight regularizer: $\sum_p |w_p|^2$
- weight normaliz° (batch-norm) $\Rightarrow$ $H \not> 0$ $\Rightarrow$ still overfit

Optimiz° landscape

- local minima:
  - global minimum $\Leftrightarrow$ θ^* particular parameters° of the network

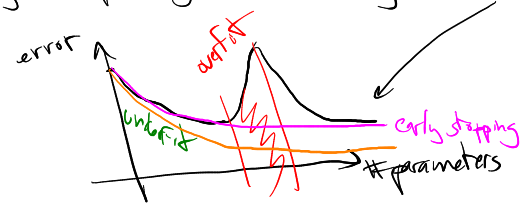


N! possible θ' for same function f_{θ^*} From permutations

  - bound on the maximal number of local minima — [Possio]
- saddle point:

Overparameteriz°

[Scaling hypoth° of generaliz° ---] "double gradient descent"



new!

- fixed architecture with variable width
- train without early stopping for a fixed # of epochs
- without regularizat°

→ train 20 models without early stopping
↳ ensemble method on top

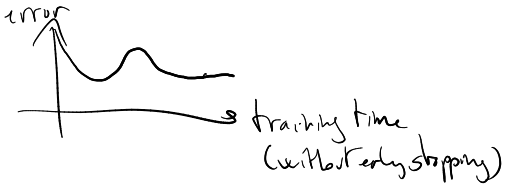
ex: Linear regression

N samples of dim d, x_i desired outputs y_i + noise

N equ° with N unknowns $\left\{ \begin{array}{l} \\ \\ \\ \end{array} \right.$ $f(x_i) = y_i$ → 1 unique solution non stable

[Deep double descent] :

ex: of ↗ dataset size ⇒ ↘ performance



⇒ take away message: put as many neurons as you can.

#6: too many neurons lead to overfit? NO!

MDL: high redundancy

→ counting #parameters: not the right way to estimate model complexity

↳ ex: compression techniques ⇒ /100 size with similar accuracy

↳ video object detection on smartphone [Wenling]