Mathematics of Learning ?

Journée Thématique Machine Learning – Lyon Probability Seminar

---

Aurélien Garivier

Jan. 23rd, 2020

Table of contents

Framework: Machine Learning

What we want to do: prediction

Phenomenon: observations $(x, y) \in \mathcal{X} \times \mathcal{Y}$ in a product of measurables spaces $\mathcal{X} \subset \mathbb{R}^p$ and $\mathcal{Y} \subset \mathbb{R}^q$.

Goal: predict y from x. Prediction error measure by *loss* $\ell(\hat{y}, y) = \|\hat{y} - y\|^2/2$ typically.

Statistical hypothesis: there exists $F : \mathcal{X} \times \Omega \to \mathcal{Y}$ such that the observations are distributed as (X, Y) where X has distribution $\mathbb{P}_X$ and $Y = F(X, \omega)$. Typically, $Y = f(X) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Examples:

- classification (OCR, image recognition, text classification, etc.)
- regression (response to a drug, weather or stock price forecast, etc.)

Target: best possible guess of Y given X: $f(X) = \mathbb{E}[Y|X]$.

Machine Learning

Mechanism of f is complex or hidden. Access to f only thru **examples** i.e. a sample $S_n = \big((X_1, Y_1), \dots, (X_n, Y_n)\big)$ of random pairs

Learning algorithm $\mathcal{A}_n : S_n \mapsto \hat{f}_n$ where $\hat{f}_n \in \mathcal{F} \subset \mathcal{Y}^{\mathcal{X}} \subset (\mathbb{R}^q)^{\mathbb{R}^p}$

$\mathcal{F}$ = **hypothesis class** = model. Example: linear regression

$$\mathcal{F} = \left\{ f_\theta : x \mapsto \left(\theta_{i,0} + \sum_{j=1}^{p} \theta_{i,j} x_j \right)_{1 \le i \le q} : \theta \in \mathcal{M}_{q,1+p}(\mathbb{R}) \right\}$$

Quality of prediction $\hat{y}$: **loss function** $\ell : \mathbb{R}^q \times \mathbb{R}^q \to \mathbb{R}_+$ e.g. $\ell(\hat{y}, y) = \frac{(\hat{y}-y)^2}{2}$

Quality of hypothesis $f \in \mathcal{F}$: **generalization error** = average loss

$$L(f) = \mathbb{E}\big[\ell(f(X), Y)\big]$$ expectation is on new observation (X,Y)

Quality of the learning algorithm $\mathcal{A}$: **risk** = average average loss

$$R_n(\mathcal{A}_n) = \mathbb{E}\left[L(\hat{f}_n)\right]$$ expectation is on sample S_n

Empirical Risk Minimization

Learning = how to find the best possible $f \in \mathcal{F}$?

$\rightarrow$ Minimize the **empirical loss = training error**

$$L_n(f) = \frac{1}{n} \sum_{k=1}^{n} \ell\big(f(X_k), Y_k\big) \qquad \text{average loss on the sample}$$

= unbiased estimator of the generalization error $L(f)$

Empirical Risk Minimizer: $\hat{f}_n \in \arg\min_{f \in \mathcal{F}} L_n(f)$

Example: linear regression with quadratic loss (dates back at least to Gauss) $\hat{f}_n = f_{\hat{\theta}_n}$ where $\hat{\theta}_n^T = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$, with

$$\mathbf{X} = \begin{pmatrix} 1 & X_1^1 & \dots & X_1^p \\ & \dots & & \\ 1 & X_n^1 & \dots & X_n^p \end{pmatrix} \text{ and } \mathbf{Y} = \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}$$

Regression by polynomials of degrees $1, 2, \dots, n-1$ $\rightarrow$ more parameters is not necessarily better, bias / variance tradeoff, Structural Risk Minimization (penalize empirical risk by model complexity)

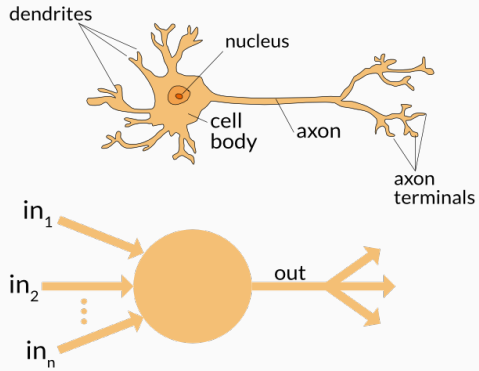
Neural Networks

Feedforward Neural Networks: Mimicking Brains?

Neuron: $x \mapsto \sigma\big(\langle w, x\rangle + b\big)$ with

- parameter $w \in \mathbb{R}^p, b \in R$
- (non-linear) activation function $\sigma : \mathbb{R} \to \mathbb{R}$

  typically $\sigma(x) = \frac{1}{1+\exp(-x)}$ or $\sigma(x) = \max(x, 0)$ called ReLU

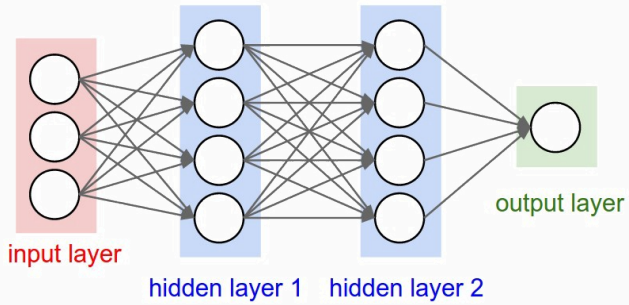


Layer: $x \mapsto \boldsymbol{\sigma}\big(Mx + \mathbf{b}\big)$ with

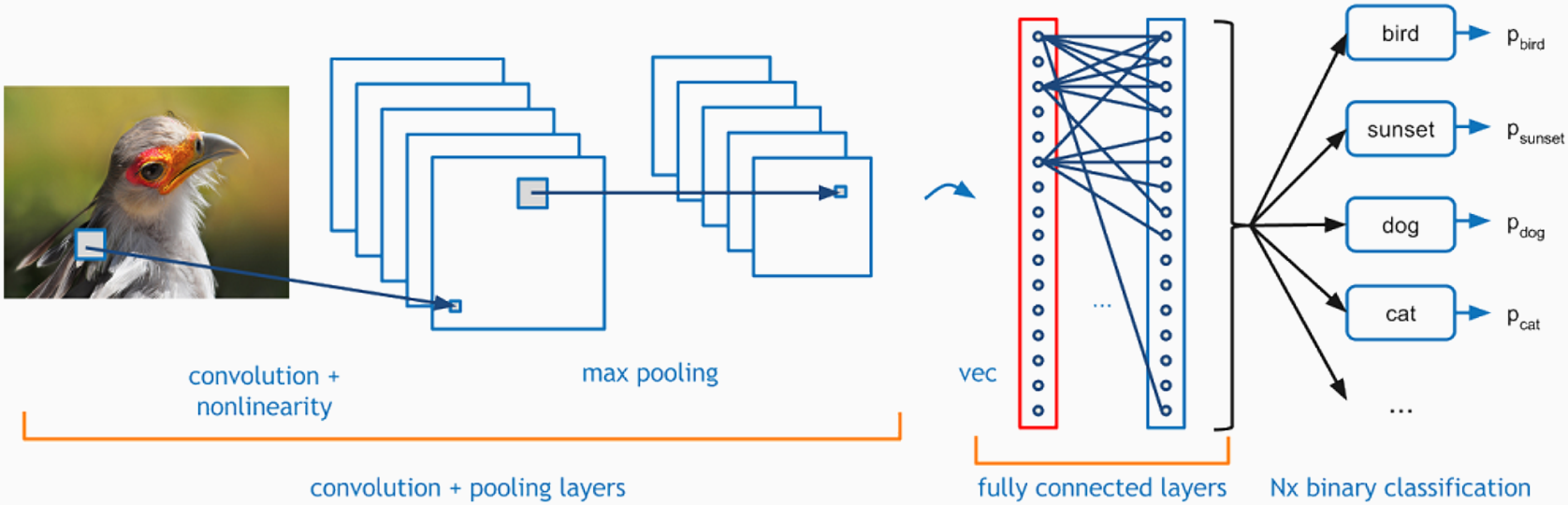
- parameter $M \in M_{q,p}(\mathbb{R}), \mathbf{b} \in \mathbb{R}^q$
- component-wise activation function $\boldsymbol{\sigma} = \sigma^{\otimes q}$

Network: composition of layers $f_\theta = \boldsymbol{\sigma}_D \circ T_D \circ \cdots \circ \boldsymbol{\sigma}_1 \circ T_1$ with

- architecture $A = \big(D, (p_1, \ldots, p_{D-1})\big)$
- $x_0 = x, \quad x_d = \boldsymbol{\sigma}_d\big(T_d x_{d-1}\big) \in \mathbb{R}^{p_d}$
- $T_d x = M_d x + \mathbf{b}_d$
- parameter $\theta = (M_1, \mathbf{b}_1, \ldots, \ldots, M_D, \mathbf{b}_D)$
  $\theta \in \Theta_A = \prod_{d=1}^{D} \mathcal{M}_{p_{d-1}, p_d}(\mathbb{R}) \times \mathbb{R}^{p_d}$
- <u>*depth*</u> D (⚠st. nb layers), <u>*width*</u> $\max_{1 \le d \le D} p_d$



Deep Neural Networks in the last Decade



Src: `https://adeshpande3.github.io/`

Several other important ideas:

- not fully connected layers
- convolution layers
- max-pooling
- dropout
- etc...

Learning with Neural Networks

How to learn with feedforward neural networks?

1. Choose architecture $A = \left[D, (p_1, \ldots, p_{D-1})\right]$
   - depth D?
   - what architectures are good if f has some with given properties?
   - activation function? sigmoid $\sigma(x) = \frac{1}{1+\exp(-x)}$ or ReLU $\sigma(x) = \max(x, 0)$

   → approximation theory?
2. Learn = find the good coefficients using S_n
   - Empirical Risk Minimization: $\hat{f}_n$ solution of

$$\min_{\substack{T_k \in \mathcal{M}_{p_d, 1+p_{d-1}}(\mathbb{R}) \\ 1 \leq d \leq D}} \frac{1}{n} \sum_{k=1}^{n} \ell\left(\boldsymbol{\sigma}_D \circ T_D \circ \cdots \circ \boldsymbol{\sigma}_1 \circ T_1(X_k), Y_k\right)$$

   - non convex, high-dimensional optimization problem
   - but gradient can be computed by **back-propagation**

   → does gradient descent work?
3. Apply $\hat{f}_n$ to new data (X, Y)
   - how to bound the generalization error $L(\hat{f}_n)$?
   - should we regularize = penalize large coefficients?

   → no overfitting?

→ How to explain the huge empirical success of deep learning?

Outline

Depth-2 Networks Are Universal

Cybenko ['89] Approximation by superposition of sigmoidal functions

Theorem

Let σ be any bounded, measurable (or continuous) function such that $\sigma(t) \to 0$ as $t \to -\infty$ and $\sigma(t) \to 1$ as $t \to \infty$. Then for every continuous function f on $[0,1]^p$ there exists a width p_1 and a depth-2 neural network f_θ with activation functions $\sigma_1 = \sigma$ and $\sigma_2 = id$

$$f_\theta(x) = \sum_{j=1}^{p_1} \alpha_j \sigma\big(\langle w_j, x\rangle + b_j\big)$$

such that $\|f_\theta - f\|_\infty$.

Proof:

- these functions σ are such that if for a measure μ on $[0,1]^p$

$$\int_{[0,1]^p} \sigma\big(\langle w, x\rangle + b\big) d\mu(x) = 0$$

  for all $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$, then $\mu = 0$.
- Hahn-Banach + Riesz representation: the closure of $\left\{ f_\theta : \theta \in \mathcal{M}_{p_1, p+1}(\mathbb{R}) \times \mathbb{R}^{p_1} \right\}$ has empty complement

An Quantitative bounds for ReLU depth-2 networks

Lemma [e.g. Eldan&Shamir'16]

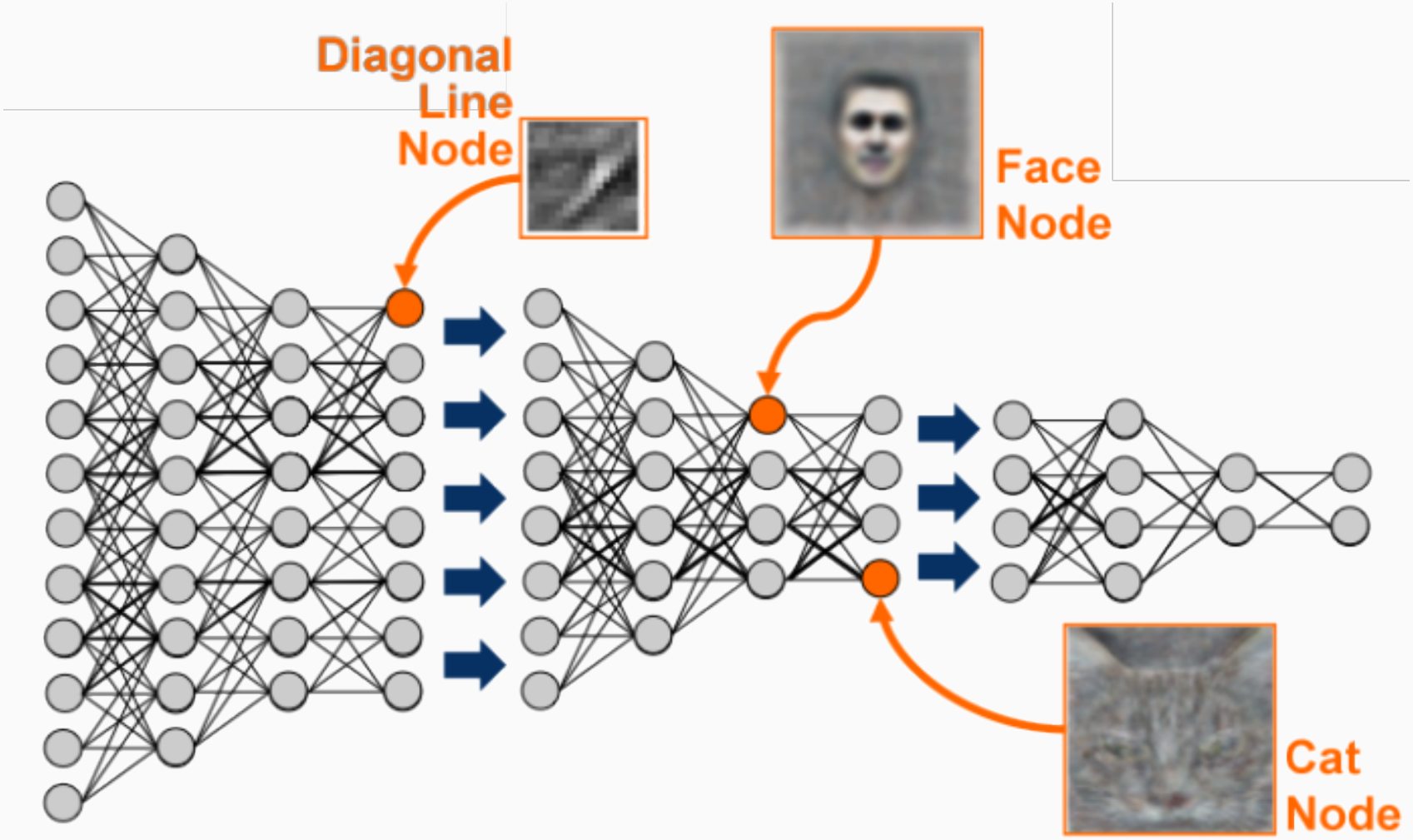
Let $g : \mathbb{R} \to \mathbb{R}$ be constant outside of an interval $[-R, R]$ and L-Lipschitz. There exists a depth-2 ReLU network f with linear output of width at most $8RL/\epsilon$ and weights at most $\max\left(2L, \|g\|_\infty\right)$ such that $\|f - g\|_\infty \leq \epsilon$.

Proof. If $2RL \leq \epsilon$, take f to be constantly equal to $g(-R)$.
Otherwise, take $m = \lceil RL/\epsilon \rceil \leq 2RL/\epsilon$, and let f be the piecewise linear function coinciding with g at points $x_i = i\epsilon/L$, $i \in \{-m, \ldots, m\}$, linear between x_i and x_{i+1}, and constant outsite of $[-x_{-m}, x_m]$. Since g is L-Lipschitz, $\|f - g\|_\infty \leq \epsilon$. But f can be written as a depth-2 ReLU network with $2m + 2 \leq 8RL/\epsilon$ neurons:

$$f(x) = f(x_{-m}) + \sum_{i=-m}^{m} \left[f'(x_i+) - f'(x_i-)\right] r(x - x_i)$$

where $f'(x_i+) = g(x_{i+1}) - g(x_i)$ and $f'(x_i-) = g(x_i) - g(x_{i-1})$ for all $-m < i < m$. Except maybe for the constant $f(x_{-m}) = g(-R)$, the coefficients are bounded by $\left|g(x_{i+1}) - g(x_i) - g(x_i) - g(x_{i-1})\right| \leq 2L$.

Why deep learning, then? The dream

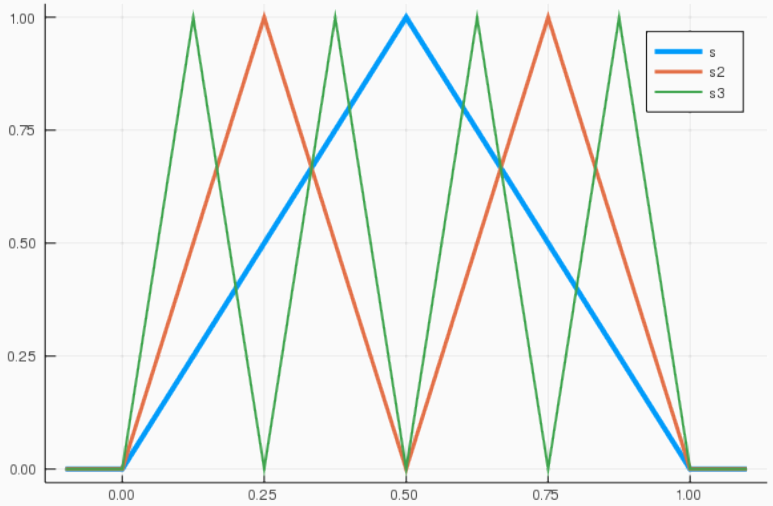


Depth is useful: sawteeth function

$$\text{Let } s(x) = \begin{cases} 2x & \text{if } 0 \leq x \leq \frac{1}{2} \\ 2 - 2x & \text{if } \frac{1}{2} \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

$$= 2r(x) - 4r\left(x - \frac{1}{2}\right) + 2r(x-1)$$

and for all $m \geq 1$ let $s_m = \underbrace{s \circ \cdots \circ s}_{m \text{ times}}$



Lemma

For all $m \geq 1$, all $k \in \{0, \ldots, 2^{m-1} - 1\}$ and all $t \in [0, 1]$,

$$s_m\left(\frac{k+t}{2^{m-1}}\right) = \begin{cases} 2t & \text{if } t \leq \frac{1}{2} \\ 2 - 2t & \text{if } t \geq \frac{1}{2} \end{cases}$$

Depth is useful: toy example

Let $g(x) = x^2$, and for $m \geq 1$ let $g_m(x)$ be such that $\forall k \in \left\{0, \ldots, 2^m\right\}$:

- $g_m\left(\frac{k}{2^m}\right) = g\left(\frac{k}{2^m}\right)$
- g_m is linear on $\left[\frac{k}{2^m}, \frac{k+1}{2^m}\right]$

Lemma

For all $k \in \left\{0, \ldots, 2^{m-1} - 1\right\}$ and all $t \in [0, 1]$,

$$g_m\left(\frac{k+t}{2^{m-1}}\right) - g\left(\frac{k+t}{2^{m-1}}\right) = \frac{t(1-t)}{4^m}$$

In particular, $\|g - g_m\|_\infty = \frac{1}{4^{m+1}}$ and for all $m \geq 2$,

$$g_m = g_{m-1} - \frac{1}{4^m}\, s_m = id - \sum_{j=1}^{m} \frac{1}{4^j}\, s_j$$

Corollary

For every $\epsilon > 0$, there exists a neural network f of depth $\lceil \log_4(1/\epsilon) \rceil$, width 3 and coefficients in $[-4, 2]$ such that $\|f - g\|_\infty \leq \epsilon$ on $[0, 1]$

Depth is useful: toy example

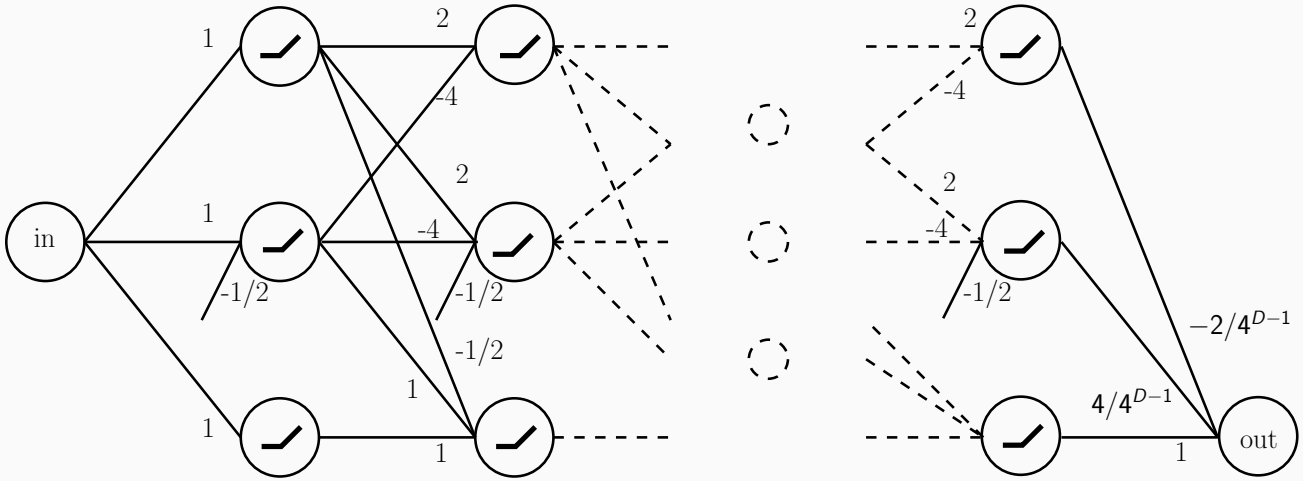
Lemma

$\|g - g_m\|_\infty = \frac{1}{4^{m+1}}$ and for all $m \geq 2$,

$$g_m = g_{m-1} - \frac{1}{4^m} s_m = id - \sum_{j=1}^{m} \frac{1}{4^j} s_j$$

Corollary

For every $\epsilon > 0$, there exists a neural network f of depth $\lceil \log_4(1/\epsilon) \rceil$, width 3 and coefficients in $[-4, 2]$ such that $\|f - g\|_\infty \leq \epsilon$ on $[0, 1]$



$x_0 = x$ $x_1 = x$ $x_2 = x - \frac{s(x)}{4}$ $x_D = x - \frac{s(x)}{4} - \cdots - \frac{s_{D-1}(x)}{4^{D-1}}$

Examples

Square on $[-1, 1]$**:** $|x| = r(x) + r(-x)$ → one additionnal width-2 layer is sufficient

Product: $\forall x, y \in \mathbb{R}$, $xy = [(x + y)^2 - (x - y)^2]/4$ → same depth, width 5

Polynomials: approximated by products

Continuous functions on $[0, 1]$**:** use uniform approximation of Lagrange interpolation at Chebishev's points [Liang & Srikant '19]

See [M. Telgarsky '16-'19. Benefits of depth in neural networks]

See work and presentation by Rémi Gribonval

Exponential separation result: [Daniely '17. Depth Separation for Neural Networks]

Outline

Computing the Gradient by Backpropagation

For every layer $d \in \{1, \dots, D\}$, we define the vector $\delta_d \in \mathbb{R}^{p_d}$ by $\delta^d(i) = \frac{\partial r}{\partial x_d(i)} \sigma'_d\big(\tilde{x}_d(i)\big)$

Recursive Equations of Backpropagation

For the squared loss $\ell(\hat{y}, y) = \frac{\|\hat{y}-y\|^2}{2}$,

$$\delta_D = \frac{1}{n} \sum_{k=1}^{n} (\hat{f}_n(X_k) - Y_k) \,.*\, \sigma'_d(\tilde{x}_D(k))$$

$$\delta_{d-1} = M_d^T \delta^d \,.*\, \sigma'_{d-1}(\tilde{x}_{d-1})$$

$$\nabla_{M_d} r = \delta_d \, x_{d-1}^T$$

Cf. Automatic Differentiation.

Gradient Descent on the empirical loss

Let $r(\theta) = L_n(f_\theta) = \frac{1}{n}\sum_{k=1}^{n} \ell\big(f_\theta(X_k), Y_k\big)$

- The weights are initialized at random, e.g. $\theta_0^d(i,j) \sim \mathcal{N}(0,1)$
- Then, they are updated by gradient descent: $\theta_t = \theta_{t-1} - \eta_t \nabla r$
- Possibility to penalize the empirical loss with $\|\theta\|^2 \rightarrow$ adds a tampering term in gradient descent
- Possibly Stochastic Gradient Descent: pick a point (or a batch) at random (or turn on the data in epochs)
- convergence to a local minimum (and how to choose η_t)?
- to a global minimum? especially when over-parameterized?
  See [Mei, Montanari, Nguyen '18-'19. A Mean Field View of the Landscape of Two-Layers Neural Networks]

Outline

Overfitting: the Double Descent Phenomenon

Src: `https://openai.com`

Classical statistics suggest that there are too many parameters wrt. the number of observations, BUT this is not what is empirically observed!

Deep neural nets overfit, but (contrary to polynomials) they seem to generalize well (especially in high dimension)

→ how to explain that?

Beginning of answer: Benign Overfitting in Linear Regression Bartlett, by Long et al., 2019