Mathematical Control and Deep Learning

Borjan Geshkovski
Universidad Autónoma de Madrid & Fundación Deusto

CAA Seminar, FAU Erlangen-Nürnberg
January 21, 2020

Universidad Autónoma de Madrid

Fundación Deusto Deustu Fundazioa

Example

Interested in approximating a relation between spatial position $x \in \mathbb{R}^2$ and altitude $y \in \mathbb{R}$.
This relation is given by continuous function $f : \mathbb{R}^2 \to \mathbb{R}$.
We don't know this function explicitly.
We know its values $\{Y_i\}_{i=1}^N \subset \mathbb{R}$ at N distinct points $\{X_i\}_{i=1}^N \subset \mathbb{R}^2$.
Machine learning consists in approximating f by:

1. Proposing a candidate approximation function $f_L(\Theta, \cdot) : \mathbb{R}^2 \to \mathbb{R}$, depending on a set of parameters Θ and some $L \geq 1$;
2. Optimizing the parameters Θ so that $f_L(\Theta, \cdot)$ fits the data $\{(X_i, Y_i)\}_{i=1}^N$ (minimize the discrepancy between $f_L(\Theta, X_i)$ and Y_i)

Example

Describe relation between any picture (vector $x \in \mathbb{R}^d$, $d \gg 1$) and the category to which it belongs ($y \in \{c_0, c_1\} \subset \mathbb{R}$).
Relation given by a function $f : \mathbb{R}^d \to \{c_0, c_1\}$.
We don't know this function explicitly.
We know its values $\{Y_i\}_{i=1}^N$ for N pictures $\{X_i\}_{i=1}^N \subset \mathbb{R}^d$.
Machine learning consists in approximating f by:

1. Proposing a candidate approximation function $f_L(\Theta, \cdot) : \mathbb{R}^d \to \mathbb{R}$, depending on a set of parameters Θ and some $L \geq 1$;
2. Optimizing the parameters Θ so that $f_L(\Theta, \cdot)$ fits the data $\{(X_i, Y_i)\}_{i=1}^N$

Machine Learning (in general)

- Interested in approximating a function $f : \mathbb{R}^d \to \mathbb{R}^m$, of some class, which we don't know explicitly.
- We have data: its values $\{Y_i\}_{i=1}^N \subset \mathbb{R}^m$ at N distinct points $\{X_i\}_{i=1}^N \subset \mathbb{R}^d$.
- Machine learning consists in approximating f by:
  1. Proposing a candidate approximation function $f_L(\Theta, \cdot) : \mathbb{R}^d \to \mathbb{R}^m$, depending on a set of parameters Θ and some $L \in \mathbb{N}$;
  2. Optimizing the parameters Θ so that $f_L(\Theta, \cdot)$ fits the data $\{(X_i, Y_i)\}_{i=1}^N$;
- Neural Networks: a way of constructing $f_L(\Theta, \cdot)$, done by L successive compositions of a specified nonlinear function σ and linear transformations depending on parameters

- Idea behind NN: approximate the unknown function f by (say $L \geq 2$) successive compositions of a specified nonlinear function and a parametrized affine transformation of the input.
- This nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$ is referred to as the *activation function*
- Examples include $\sigma(x) = \max\{0, x\}$ ("ReLU") and $\sigma(x) = \frac{1}{1+e^{-x}}$ ("sigmoid")
- σ generally applied to vectors in $\mathbb{R}^d$, so meant component-wise

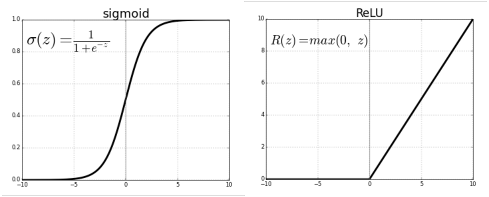

Figure: The *sigmoid* and *ReLU* activation functions.

Neural Networks

Definition (Neural network)

A *neural network* of *depth* $L \geq 2$, with *input* dimension $d \geq 1$, and *output* dimension $m \geq 1$, is a tuple

$$\Theta = \{(A_k, b_k)\}_{k=1}^{L}$$

of matrix-vector pairs, where

$$A_k \in \mathbb{R}^{N_k \times N_{k-1}} \quad \text{and} \quad b_k \in \mathbb{R}^{N_k} \quad \text{for } k = 1, \ldots, L.$$

The numbers $\{N_k\}_{k=0}^{L} \in \mathbb{N}$ with $N_0 = d$ and $N_L = m$ are given, and called *widths*.

- The parameters (A_k, b_k) are called *weights* and *biases*.
- Sometimes $N_k = d$ for all $k = 0, \ldots, L$ (*ResNets*)

Let $\sigma \in C^0(\mathbb{R})$ be a fixed activation function.

Definition (Layers)

Let $\Theta = \{(A_k, b_k)\}_{k=1}^{L}$ be a neural network of depth $L \geq 2$, with input dimension $d \geq 1$, and output dimension $m \geq 1$.
For $k = 1, \ldots, L$ define the affine map

$$\begin{aligned} \Lambda_k : \mathbb{R}^{N_{k-1}} &\longrightarrow \mathbb{R}^{N_k} \\ x &\longmapsto A_k x + b_k. \end{aligned}$$

We call $\sigma(\Lambda_k(\cdot))$ the k^{th}-*layer* of the neural network Θ.

- $\mathrm{Id}_{\mathbb{R}^d}$ is the *input layer*.
- $\sigma(\Lambda_1), \ldots, \sigma(\Lambda_{L-1})$ are the *hidden* layers.
- $\sigma(\Lambda_L)$ is the *output* layer.
- $L = 2 \longrightarrow$ *shallow* network; $L \geq 3 \longrightarrow$ *deep* network.

Neural Networks

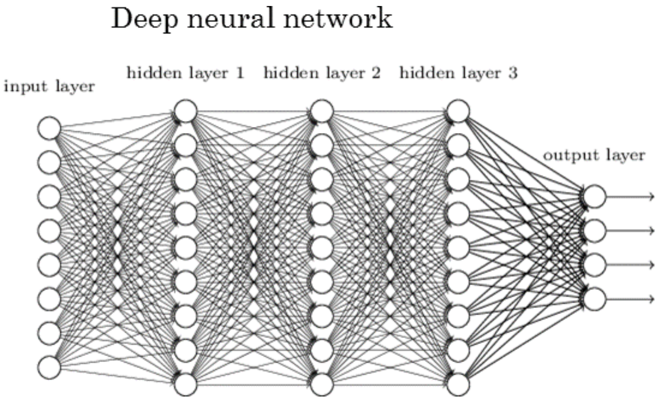

Figure: Graphical representation of a neural network of depth $L = 4$. The input layer has $d = 8$ nodes (one for each component of a data point $X_i \in \mathbb{R}^d$), the output layer has $m = 4$ nodes; the width of the hidden layers equals 9.

Definition (Realization of a neural network)

Let $\sigma \in C^0(\mathbb{R})$ be fixed activation function.
Let $\Theta = \{(A_k, b_k)\}_{k=1}^L$ be a neural network of depth $L \geq 2$, with input dimension $d \geq 1$, and output dimension $m \geq 1$.
The *realization* of Θ w.r.t. σ is the function

$$\begin{aligned} f_L(\Theta, \cdot) \colon \mathbb{R}^d &\longrightarrow \mathbb{R}^m \\ x &\longmapsto \sigma(\Lambda_L\left(\sigma(\Lambda_{L-1}(\sigma \circ \cdots \circ \sigma(\Lambda_1(x))))\right)). \end{aligned}$$

- $f_L(\Theta, \cdot)$ depends on the *depth* $L \geq 1$, and the *width* $\max_k N_k \geq 1$;
- Oftentimes the realization $f_L(\Theta, \cdot)$ itself is called neural network.

Universal approximation theorem(s)

Theorems generally of the form: *The class of neural networks is dense with respect to some topology in some function class $\mathcal{C}$.*

Theorem (Cybenko '89, MCSS)

Let $\sigma : \mathbb{R} \to \mathbb{R}$ *be a nonconstant, bounded and continuous function. Let* $d \geq 1$*. Given any* $\varepsilon > 0$ *and any* $f \in C^0([0,1]^d)$*, there exists* $m \in \mathbb{N}$*, constants* $v_k, b_k \in \mathbb{R}$ *and vectors* $A_k \in \mathbb{R}^d$ *for* $k = 1, \ldots, m$ *such that* F *defined by*

$$F(x) = \sum_{k=1}^{m} v_k \sigma(A_k^T x + b_k)$$

satisfies

$$\sup_{x \in [0,1]^d} |f(x) - F(x)| < \varepsilon.$$

Ingredients of the proof: Contradiction argument + Hahn Banach + Riesz-Representation + properties of σ.

Universal approximation theorem(s)

More recent results for ReLU networks include

Theorem (Hanin '17)

Let $d \geq 1$ and let $f : [0,1]^d \to \mathbb{R}$ be a positive and continuous function with $\|f\|_\infty = 1$. Then for any $\varepsilon > 0$, there exists a realization g of a ReLU network of depth

$$L = \frac{2\,d!}{w_f(\varepsilon)^d}$$

and width $\max_k N_k \leq d+3$ such that

$$\|f-g\|_\infty \leq \varepsilon.$$

Here $w_f : \delta \mapsto \sup\{|f(x)-f(y)| \colon |x-y| \leq \delta\}$ denotes the modulus of continuity of f.

Improved results for functions $f : \mathbb{R}^d \to \mathbb{R}^m$ can be found in Müller '20. The proofs are more constructive than Cybenko '89.

Training phase (Optimization)

- We are given data $\{(X_i, Y_i)\}_{j=1}^N \subset \mathbb{R}^d \times \mathbb{R}^m$ (training set);
- Training consists in solving the optimization problem:

$$\min_{\Theta=\{(A_k,b_k)\}_{k=1}^L} \sum_{i=1}^N |Y_i - f_L(\Theta, X_i)|^2 + \alpha\mathcal{R}(\Theta); \tag{1}$$

$\alpha > 0$ is a regularization parameter, $\mathcal{R}$ convex;

- Non-convex optimization problem because of f_L;
- Existence of a minimizer may be shown by a direct method ($\sigma \in C^0$);

Once training is done:

- Given a minimizer $\widehat{\Theta}$, we set $f(x) := f_L(\widehat{\Theta}, x)$ (*regression*) or

$$f(x) := \begin{cases} c_0 & \text{if } f_L(\widehat{\Theta}, x) \leq \dfrac{c_0 + c_1}{2} \\ c_1 & \text{else} \end{cases}$$

(*classification*).

Computing the minimizer

The functional to be minimized is of the form

$$J(\Theta) = \sum_{i=1}^{N} J_i(\Theta). \tag{2}$$

We could do gradient descent:

$$\Theta^{n+1} := \Theta^n - \eta \nabla J(\Theta^n),$$

η is step-size. But often $N \gg 1$.

- Stochastic gradient descent:
    1. pick $i \in \{1, \ldots, N\}$ uniformly at random
    2. $\Theta^{n+1} := \Theta^n - \eta \nabla J_i(\Theta^n)$
- Use adjoints to compute these gradients ("backpropagation")
- Issues: might not converge to global minimizer; also how does one initialize the weights in the iteration?

A discretized dynamical system

Recall the realization of the neural network $\Theta = \{(A_k, b_k)\}_{k=1}^{L}$:

$$f_L(\Theta, x) := \sigma(\Lambda_L(\sigma \circ \ldots \circ \sigma(\Lambda_1(x)))).$$

We can define a scheme: for $x \in \mathbb{R}^d$,

$$\begin{cases} Z^{k+1} &= \sigma(A_{k+1} Z^k + b_{k+1}) \quad \text{for } k = 0, \ldots, L-1 \\ Z^0 &= x \end{cases} \tag{3}$$

- We recognize a discrete-time dynamical system;
- Training can thus be rewritten as a constrained optimization problem:

$$\min_{\Theta = \{(A_k, b_k)\}_{k=1}^{L}} \sum_{i=1}^{N} |Y_i - Z^L|^2 + \alpha \mathcal{R}(\Theta)$$

  with $Z^L = Z^L(\Theta, X_i)$, subject to (3) with $Z^0 = X_i$.
- In this case, deep learning may be seen as discretized optimal control (the parameters Θ play the role of controls).

Residual Neural Networks (ResNets)

Back to neural networks.
A different *architecture* (He et al. '15, Weinan E et al. '17, '18, '19):

$$\begin{cases} Z^{k+1} &= Z^k + \Delta t\, \sigma(A_k Z^k + b^k) \quad \text{for } k = 1, \ldots L-1 \\ Z^0 &= x \in \mathbb{R}^d, \end{cases} \tag{4}$$

with $\Delta t = \frac{T}{L}$ and $T > 0$ given time horizon.

- requires uniform widths $N_k = d$ at each layer k (can be prohibitive for applications)
- Recognize explicit Euler scheme for ODE

$$\begin{cases} z'(t) = \sigma(A(t)z(t) + b(t)) \quad \text{for } t \in (0, T) \\ z(0) = x \in \mathbb{R}^d. \end{cases} \tag{5}$$

- The limit $L \to +\infty$ is treated in Thorpe et al. '19.

The continuous Optimal Control Problem

- In view of what precedes, can be advantageous (lightens the notations) to consider the continuous-time optimal control problem:

$$\inf_{u(t)\in U} \sum_{i=1}^{N} |Y_i - z(T)|^2 + \alpha\mathcal{R}(u) \tag{6}$$

subject to

$$\begin{cases} z'(t) & = F(u(t), z(t)) \quad \text{in } (0,T) \\ z(0) & = X_i \in \mathbb{R}^d. \end{cases}$$

- We wrote $u(t) = (A(t), b(t))$ and $F(u,z) = \sigma(Az+b)$.
- σ globally Lipschitz; existence of a minimizer is more tricky (see Trélat '05); here $U \subset L^\infty(0,T;\mathbb{R}^d)$
- An example: $\mathcal{R}(u) = \int_0^T \ell(z(t), u(t))dt$.
- Easier to write optimality system (E et al. '18), can use different algorithms for training (shooting method);
- Other schemes for ODE to obtain new architectures (Runge-Kutta)

Questions, perspectives

Many questions persist:

- How can one quantify/describe the stability of the deep learning process with respect to perturbations in the data?
- What about the choice of the activation function σ? Can, depending on the application, a better σ be deduced as the nonlinearity from a PDE?
- How does one best choose the length L and widths N_k of the neural network?
- Can ResNets formulated with non-uniform widths N_k?
- What mathematical control results does one transfer to deep learning?
- Many other architectures were not presented (Convolutional Neural Networks..)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 765579.

Thank you for your attention.

This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No 694126 - DyCon)