Control in Deep Learning and Neural Networks[1]

Enrique Zuazua
enrique.zuazua@fau.de

FAU - AvH

April 13, 2020

[1] This material was provided by Borjan Geshovski and is part of his PhD Thesis planned for 2021 at UAM-Madrid

What is Machine Learning?

A simplified map of Machine Learning

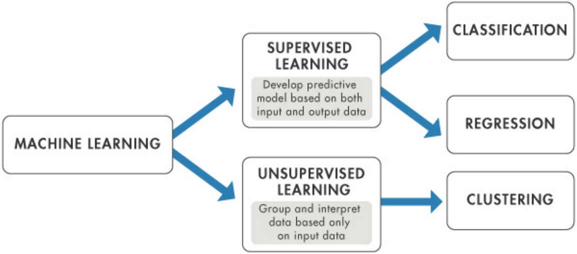

MACHINE LEARNING
SUPERVISED LEARNING
Develop predictive model based on both input and output data
UNSUPERVISED LEARNING
Group and interpret data based only on input data
CLASSIFICATION
REGRESSION
CLUSTERING

Figure: Source: https://de.mathworks.com/help/stats/machine-learning-in-matlab.html

Machine Learning (Supervised learning)

- Interested in approximating a function $f : \mathbb{R}^d \to \mathbb{R}^m$, of some class $(C^0, L^1 \dots)$, which we don't know explicitly.
- We have data: its values[2] $\{\vec{y}_i\}_{i=1}^S \in (\mathbb{R}^m)^S$ at S distinct points $\{\vec{x}_i\}_{i=1}^S \in (\mathbb{R}^d)^S$.
- Generally split the S data points into *training* data $\{\vec{x}_i, \vec{y}_i\}_{i=1}^N$ and *testing* data $\{\vec{x}_i, \vec{y}_i\}_{i=N+1}^S$.
- Machine learning consists in:
  1. Propose a candidate approximation $f_\Theta(\cdot) : \mathbb{R}^d \to \mathbb{R}^m$, depending on parameters Θ and some hyper-parameters $L \geq 1$, $\{N_k\}$;
  2. Tune Θ as to minimize the *empirical risk*[3]

$$\sum_{i=1}^N \ell(f_\Theta(\vec{x}_i), \vec{y}_i),$$

     where $\ell \geq 0$, $\ell(x, x) = 0$ (e.g. $\ell(x, y) = |x - y|^2$). This is called *training*.
  3. Check if test error $\sum_{i=N+1}^S \ell(f_\Theta(\vec{x}_i), \vec{y}_i)$ is small. This is called *generalization*.

[2] Possibly noisy, so $f(\vec{x}_i) = \vec{y}_i + \varepsilon_i$ with $\varepsilon_i \sim \mathcal{N}(0, \varsigma^2)$ for instance.

[3] Also called *training error*

Model validation

Question: After training, how do we know if the optimized neural net is good?
Answer: Check if the test error $\sum_{i=N+1}^{S} \ell(f_\Theta(\vec{x}_i), \vec{y}_i)$ is small.

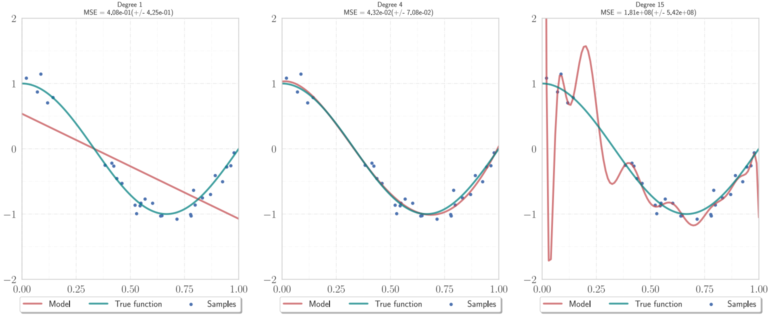


Figure: *Underfitting*, *good generalization*, and *overfitting*. We wish to recover the function $f(x) = \cos(\frac{3}{2}\pi x)$ (blue) on $(0, 1)$ from $S = 20$ noisy data samples. Constructed approximations using $N = 12$ training data, while the remaining 8 samples are used for testing the results. The most complicated model (right) is not necessarily the best (Occam's razor) This is related to *the Runge phenomenon*.

Supervised learning

Example (Binary classification)

- Assume that all the points in some $\Omega \subset \mathbb{R}^d$ are either blue or red in a not too incoherent way.
- **Goal**: Given an arbitrary blue or red point $\vec{x} \in \Omega \subset \mathbb{R}^d$, say what is the color of $\vec{x}$ (for simplicity, 0 is blue and 1 is red).
- Relation point-color represented by a function $f : \mathbb{R}^d \to \{0, 1\}$, which we don't know explicitly.
- **We know the colors** $\{\vec{y}_i\}_{i=1}^M \in \{0,1\}^M$ **of** M **points** $\{\vec{x}_i\}_{i=1}^M \in (\mathbb{R}^d)^M$. We take N training data.
- **Machine learning** consists in:
  1. Proposing a candidate approximation $f_\Theta(;\cdot) : \mathbb{R}^d \to \mathbb{R}$, depending on parameters Θ and some fixed hyper-parameters $L \geq 0$, $\{N_k\}_{k=0}^L$;
  2. Optimizing Θ so that $\sum_{i=1}^N |f_\Theta(;\vec{x}_i) - \vec{y}_i|^2$ is small
  3. Compute optimizer $\widehat{\Theta}$; then given any $\vec{x} \in \Omega$,

     $$F(\vec{x}) := \mathbb{1}_{\left\{f_L(\widehat{\Theta};\cdot) \geq \frac{1}{2}\right\}}(\vec{x})$$

     will yield the color.

The (simplified) example

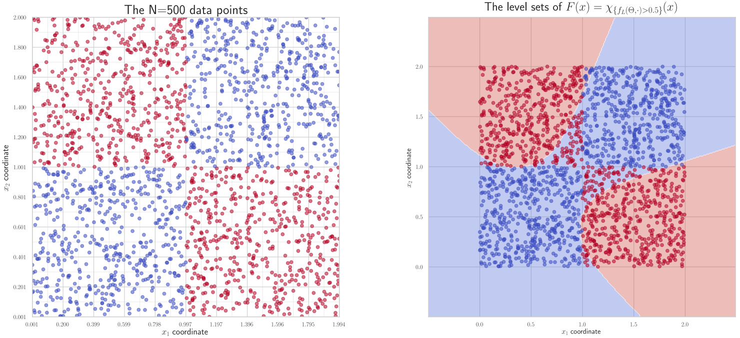


Figure: We have the training data points $\{\vec{x}_i\}_{i=1}^{4000} \in ([0,2]^2)^{4000}$ on the left, each having respective colors $\{y_i\}_{i=1}^{4000} \in \{0,1\}^{4000}$. The data are arranged in a chess-like pattern. On the right, we plot the level-sets of $F(\vec{x}) := \mathbb{1}_{\left\{f_L(\widehat{\Theta};\cdot)\geq\frac{1}{2}\right\}}(\vec{x})$ for $\vec{x} \in [-0.5, 2.5]^2$ which separates the points with 80% accuracy. We used a neural network with 1 hidden layer and 5 components.

A motivating practical example

Remark: This classification procedure can be used for classification in image processing (e.g. cat-dog recognition, handwritten digit recognition).

Figure: Classifying handwritten digits from the MNIST dataset using a neural network. The inputs $\{\vec{x}_i\}_{i=1}^{50} \in (\mathbb{R}^{28\times 28})^{50}$ are pixelated images, while the labels $\{\vec{y}_i\} \in (\mathbb{R}^{10})^{50}$ correspond to numbers from 0 to 9, each one identified with an element of the canonical basis $\{e_k\}_{k=1}^{10}$ of $\mathbb{R}^{10}$. We used a network with 2 hidden layers (one with 256 and a second with 64 neurons). Illustrated are the transitions through the layers when the datum is the digit $5 \simeq e_6$. The three rightmost plots indicate the three layers of the model, namely the middle 2 are the values of $\sigma(A^0 z^0 + b^0) \in [0,1]^{256}$ and $\sigma(A^1 z^1 + b^1) \in [0,1]^{64}$ (reshaped in a matrix). The brighter a pixel, the closer the value is to 1

Example ("Learning" $\sin(x)$)

- **Goal**: Given $M = 4000$ noisy values of $\sin(x)$ on $(0, 6\pi)$, deduce the original function $\sin(x)$.
- **We thus know the values** $\{y_i\}_{i=1}^M \in [0,1]^M$ **of** M **points** $\{x_i\}_{i=1}^M \in (0, 6\pi)^M$. We take N training data.
- **Machine learning** consists in:
  1. Proposing a candidate approximation $f_\Theta(;\cdot) : \mathbb{R}^d \to \mathbb{R}$, depending on parameters Θ and some fixed hyper-parameters $L \geq 0$, $\{N_k\}_{k=0}^L$;
  2. Optimizing Θ so that $\sum_{i=1}^N |f_\Theta(;x_i) - y_i|^2$ is small
  3. Compute optimizer $\widehat{\Theta}$; then given any $\vec{x} \in (0,1)$, $f_L(\widehat{\Theta};\cdot)$ should approximate $f(x) = \sin(x)$.

- **Remark:** The procedure can be used to approximate more complicated functions (including solutions of PDEs, etc.)
- Thus the difference between classification and regression is in the co-domain of the function we wish to approximate. In classification, the co-domain is discrete, contrary to regression, where the function has continuous values.

The example

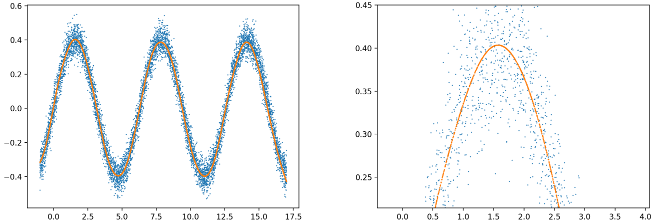


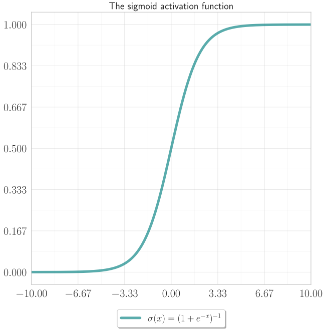
Figure: The N training data in blue and the trained neural network solution in orange (left), with a zoom in a subinterval (right). We used a neural network with 2 hidden layers consisting of 4 components each.

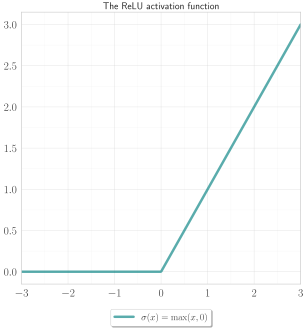
Neural networks

Activation function

Given $\sigma \in C^0(\mathbb{R};\mathbb{R})$, we **denote/define componentwise** for $z \in \mathbb{R}^d$,

$$\sigma(z) := \begin{bmatrix} \sigma(z_1) \\ \sigma(z_2) \\ \vdots \\ \sigma(z_d) \end{bmatrix} \in \mathbb{R}^d. \tag{1}$$





Neural Networks (basic definition)

Definition (Neural network)

Let $L \geq 0$, $d \geq 1$, $m \geq 1$ and $\{N_k\}_{k=1}^{L} \in \mathbb{N}^L$ be given, with $N_0 := d$ and $N_{L+1} := m$.
A **neural network with L hidden layers**, *input* dimension d, and *output* dimension m, **is a parameter-dependent map**

$$\begin{aligned} f_{\Theta}(;\cdot) : \mathbb{R}^d &\longrightarrow \mathbb{R}^m \\ \vec{x} &\longmapsto (\varphi \circ \Lambda_{L+1} \circ \sigma \circ \Lambda_L \circ \sigma \circ \ldots \circ \sigma \circ \Lambda_1)(\vec{x}) \end{aligned}$$

where $\Lambda_{k+1}\vec{z} := A^k\vec{z} + b^k$ for all $k \in \{0, \ldots, L\}$ and $\vec{z} \in \mathbb{R}^{N_k}$. The sequence of matrix-vector pairs

$$\Theta = \left\{(A^k, b^k)\right\}_{k=0}^{L}$$

where

$$A^k \in \mathbb{R}^{N_{k+1} \times N_k} \quad \text{and} \quad b^k \in \mathbb{R}^{N_{k+1}} \quad \text{for } k \in \{0, \ldots, L\}$$

are parameters, while $\sigma \in C^0(\mathbb{R})$ and $\varphi \in C^0(\mathbb{R})$ are two fixed functions.

Comment: Definition of f_L hard to read; can be rewritten equivalently as in Slide 15 below.

Some comments

Cleaner definition

Consider setting of Slide 14. Given a data point $\vec{x}_i \in \mathbb{R}^d$ and parameters Θ, a neural network writes as

$$f_\Theta(;\vec{x}) = \varphi(A^L z^L + b^L) \tag{2}$$

where $z^L = z_i^L \in \mathbb{R}^m$ being given by the scheme

$$\begin{cases} z^{k+1} = \sigma(A^k z^k + b^k) & \text{for } k = 0, \ldots, L-1 \\ z^0 = \vec{x}_i \in \mathbb{R}^d. \end{cases} \tag{3}$$

This works for $L \geq 1$. Observe that $z^k \in \mathbb{R}^{N_k}$ for $k \in \{0, \ldots, L\}$.

- We can thus define a neural network equivalently by means of (**??**)-(**??**).
- (**??**) is often called the *architecture* of the neural network
- In principle, we could add different terms in the scheme (**??**) to obtain other architectures (Slide 49).
- A neural network with $L = 1$ is called a one hidden layer (or shallow) network.

Some comments

- $\sigma \in C^0(\mathbb{R})$ generally as Slide 13 (+monotonically increasing)
- Λ_{L+1} projects data to arrival space $\mathbb{R}^m$, while φ can compress it within a desired interval (usually $[0,1]$, to interpret as probabilities).
- In binary classification (i.e. data $\{\vec{y}_i\}_{i=1}^N \in \{0,1\}^N$ like our red-blue example of Slide 7), we have $\varphi \equiv \sigma$.

  In this case, a network with 1 (hidden) layer reads

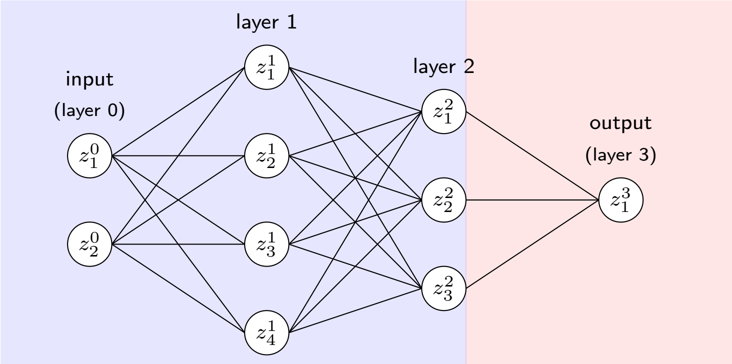
$$f(\Theta,\cdot) : \vec{x} \mapsto \sigma(A^1\sigma(A^0\vec{x}+b^0)+b^1).$$

- Otherwise, depends on the application:
  - for regression, we use $\varphi(\vec{x}) = \vec{x}$, with $\vec{x} \in \mathbb{R}^m$ (no constraints on arrival set).
  - for $m > 2$ colors we use

$$\varphi(\vec{x}) = \frac{1}{\sum_{j=1}^m \exp(x_j)} \begin{bmatrix} \exp(x_1) \\ \vdots \\ \exp(x_m) \end{bmatrix} \in \mathbb{R}^m.$$

Let us elaborate. We want to classify 3 colors (for instance blue, red and green), and we have data of the form $\{\vec{x}_i\}_{i=1}^N \in (\mathbb{R}^d)^N$. The corresponding labels are $\{\vec{y}_i\}_{i=1}^N \in (\mathbb{R}^3)^N$ as we identify an element of the canonical basis $\{e_1, e_2, e_3\}$ of $\mathbb{R}^3$ with each of the 3 colors. The function φ will take the output $z^L \in \mathbb{R}^3$ of the last hidden layer of the network, and normalizes it into a probability distribution consisting of 3 probabilities proportional to the exponentials of $\{z_j^L\}_{j=1}^3$ (i.e. $\varphi(z^L) \in (0,1)^3$). Such φ is called *softmax*.

"Discrete dynamics" of a single datum



$$\mathbb{R}^2 \quad \longmapsto \quad \mathbb{R}^4 \quad \longmapsto \quad \mathbb{R}^3 \quad \longmapsto \quad \mathbb{R}$$

$$z^0 = \vec{x} \quad \xrightarrow{\sigma \circ \Lambda_1} \quad z^1 \quad \xrightarrow{\sigma \circ \Lambda_2} \quad z^2 \quad \xrightarrow{\varphi \circ \Lambda_3} \quad z^3 = f_L(\vec{x})$$

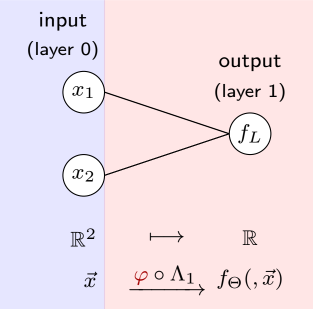
Here $L = 2$, input $d = 2$ and output $m = 1$. Hidden layers are the layers 1 and 2. The width of the layers j ($0 \leq j \leq 3$) are $N_0 = 2$, $N_1 = 4$, $N_2 = 3$, $N_3 = 1$, and $z^j = (z_1^j, \ldots, z_{N_j}^j)^{\mathsf{T}}$.

Example (no hidden layers)

Let $L = 0$ (**zero hidden layers**), $d = 2$ and $m = 1$. Then

$$f_\Theta(,\vec{x}) = \varphi\left(\sum_{j=1}^{2} A_j^0\, x_j + b^0\right) \in \mathbb{R}$$

for $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$, where $A^0 = [A_1^0 \quad A_2^0] \in \mathbb{R}^{1\times 2}$ (1 row 2 columns) and $b^0 \in \mathbb{R}$.


input
(layer 0)
output
(layer 1)
x_1
f_L
x_2
$\mathbb{R}^2 \longmapsto \mathbb{R}$
$\vec{x} \xrightarrow{\varphi \circ \Lambda_1} f_\Theta(,\vec{x})$

Examples

Let $L = 0$. Then $f_\Theta(, \vec{x}) = \varphi(A^0\vec{x} + b^0)$ for $\vec{x} \in \mathbb{R}^d$ with $A^0 \in \mathbb{R}^{m\times d}$ and $b \in \mathbb{R}^m$. Recall by convention that

$$\varphi(A_0\vec{x} + b_0) := \begin{bmatrix} \varphi\left(\sum_{j=1}^d A^0_{1,j}x_j + b^0_1\right) \\ \varphi\left(\sum_{j=1}^d A^0_{2,j}x_j + b^0_2\right) \\ \vdots \\ \varphi\left(\sum_{j=1}^d A^0_{m,j}x_j + b^0_m\right) \end{bmatrix} \in \mathbb{R}^m.$$

Example 1: The data

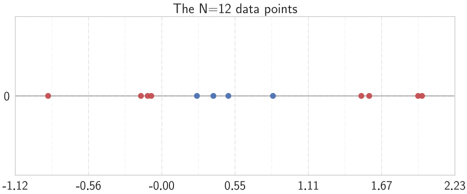


Figure: We consider 4 blue and 8 red data points.

- **Objective**: *predict the colors of other points in the interval* $\longrightarrow$ separate the interval in a set of blue and a set of red points, using the data.
- We consider a (trained) one hidden layer network with one component:

$$f_1(\Theta, x) = \sigma(A^1\sigma(A^0x + b^0) + b^1) \in \mathbb{R} \qquad \text{for } x \in \mathbb{R} \tag{4}$$

where $(A^0, A^1, b^0, b^1) \in \mathbb{R}$ are all scalars.

- In all examples, $\sigma(x) = \frac{1}{1+e^{-x}}$ (see Slide 13).

Example 1: Steps 1 & 2 - Stretch, then compress

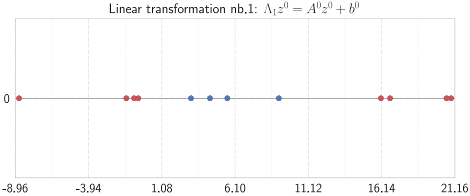


Figure: Step 1: First apply $\Lambda_1 x = A^0 x_i + b^0 \in \mathbb{R}$ (dilating and translating) to each x_i

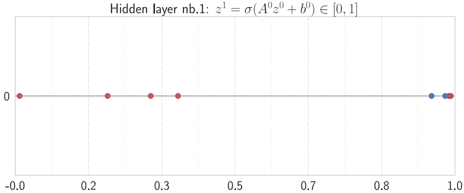


Figure: Step 2: Then apply σ to Step 1 to compress in $[0,1]$ (first try to obtain a separation)

Example 1: Steps 3 & 4 - Stretch, compress again

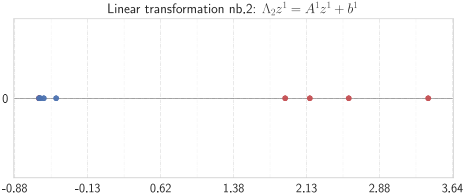


Figure: Step 3: Stretch again by $\Lambda_1 z^1 = A^1 z^1 + b^1 \in \mathbb{R}$

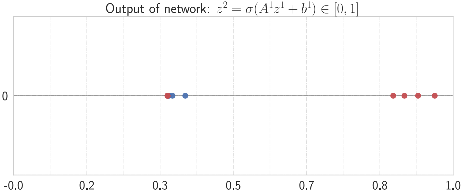
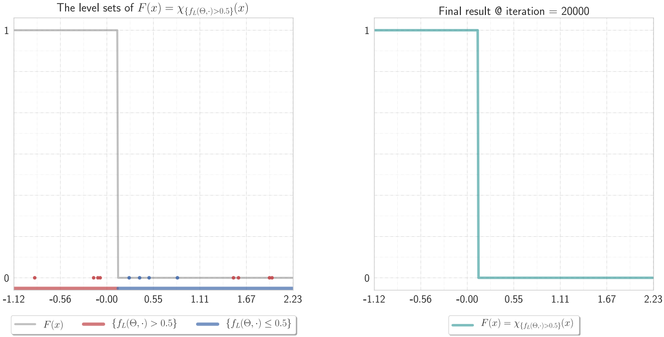


Figure: Compress Step 3 by σ.

Step 5: The final result



The level sets of $F(x) = \chi_{\{f_L(\Theta,\cdot)>0.5\}}(x)$

Final result @ iteration = 20000

Figure: On the right, we have the function which takes any point in the interval and maps to 0 (blue) or 1 (red). On the left, we see its level sets. There was not "enough room" to separate the points.

Example 2: Same data, 2 components in layer

- 1 component in the hidden layer (i.e. $A^0 \in \mathbb{R}$) doesn't separate well.
- How about 2:

$$f_1(\Theta, x) = \sigma(A^1 \sigma(A^0 x + b^0) + b^1) \in \mathbb{R} \qquad \text{for } x \in \mathbb{R} \tag{5}$$

but now $A^0 \in \mathbb{R}^{2\times 1}, A^1 \in \mathbb{R}^{1\times 2}$, and $b^0 \in \mathbb{R}^2, b^1 \in \mathbb{R}$.

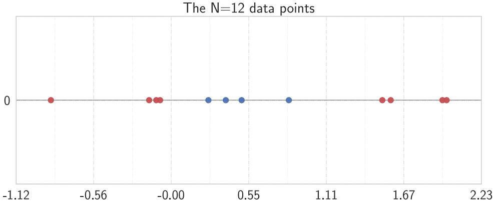


Figure: We consider the same 4 blue and 8 red data points.

Example 2: Step 1 & 2 - Embed data in $\mathbb{R}^2$, then "curb" it

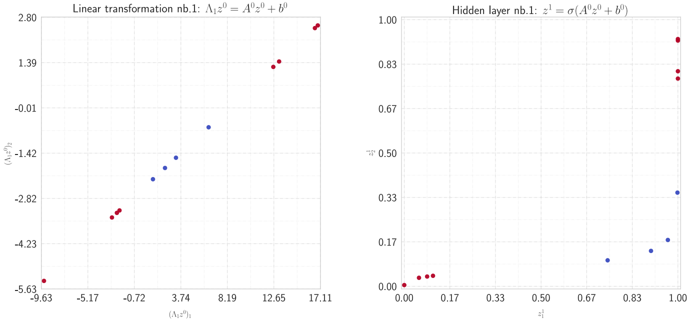


Figure: Embed the data in $\mathbb{R}^2$ via $\Lambda_1 : \mathbb{R} \to \mathbb{R}^2$ as $A^0 \in \mathbb{R}^{2\times1}$; dilates and translates the points along a diagonal in $\mathbb{R}^2$. Applying σ "curbs" the points (nonlinearity and monotonicity is important here), making it easier to separate by a line.

Example 2: Separate the data with a line

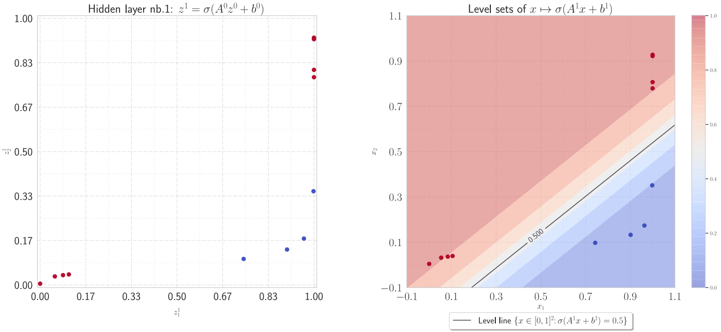


Figure: We see how the projection $x \mapsto A^1 x + b^1$ onto $\mathbb{R}$ and compression by σ would give separation (see right). Important part was done in the slide before, as working in $\mathbb{R}^2$ means that we can separate by a line.

Example 2: Step 3 & 4 - Project and compress

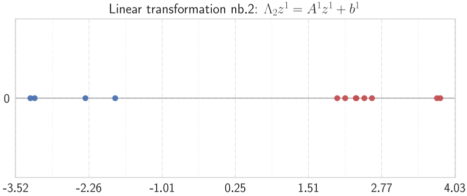


Figure: We project onto $\mathbb{R}$; the angle allows to switch the order.

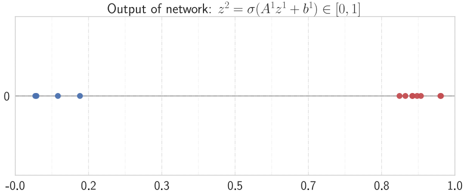


Figure: Here σ is only used as a compression to remain in $[0,1]$ (obtain a probability)

Example 2: The final result

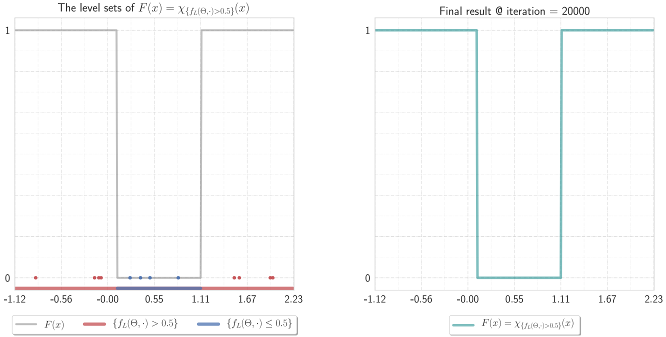


Figure: We see the level sets (left) of the classifier function (right). The function $f_1(x) = \sigma(A^1\sigma(A^0x + b^0) + b^1)$ is with values in $[0, 1]$, can be seen as probability distribution; to obtain clear-cut $0 - 1$ values, we removed the transition zone.

Example 3: 3 components in layer

We try adding another component in the hidden layer, i.e. take $A^0 \in \mathbb{R}^{3\times 1}$.

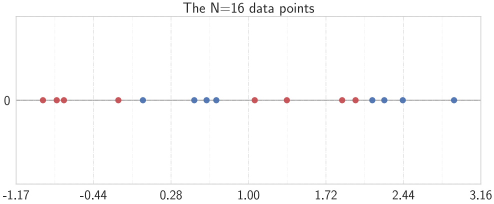


Figure: We add a component to the hidden layer. Means that the data will be embedded in $\mathbb{R}^3$ after the first linear transformation.

Example 3: Step 1 - Embed in $\mathbb{R}^3$

Linear transformation nb.1: $\Lambda_1 z^0 = A^0 z^0 + b^0$

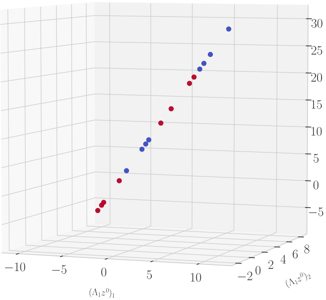


Figure: The linear transformation stretches the data onto a diagonal line in $\mathbb{R}^3$.

Example 3: Step 2 - Apply σ to curb points

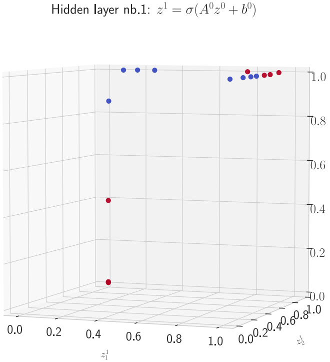


Figure: We apply the nonlinearity σ to each component, to the effect of curbing the data (as well as compressing within $[0, 1]^3$).

Example 3: We can separate in $\mathbb{R}^3$ with a hyperplane

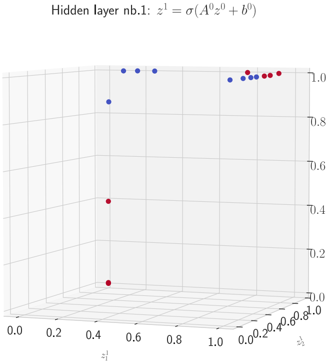


Figure: The curbed points may be separated by a hyperplane ($\{x \in \mathbb{R}^3 : \sigma(A^1 x + b^1) = 0.5\}$ in blue), much like the simpler data of Example 2 can be separated by a line. We see this on the right.

Example 3: Steps 3 & 4 - Project and compress

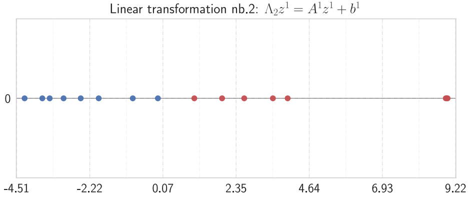
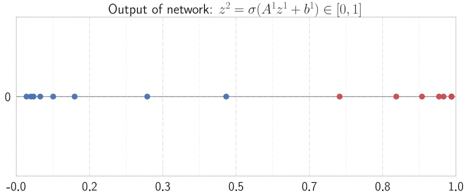
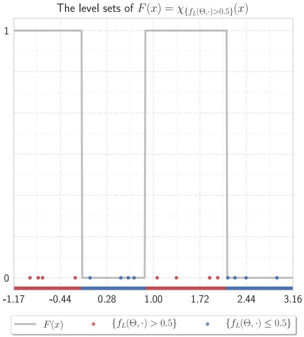
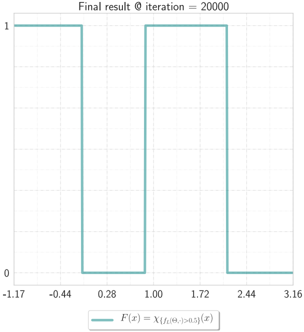

Linear transformation nb.2: $\Lambda_2 z^1 = A^1 z^1 + b^1$

Figure: We see that the data is correctly separated after the projection onto $\mathbb{R}$.


Output of network: $z^2 = \sigma(A^1 z^1 + b^1) \in [0, 1]$

Example 3: Final classifier


The level sets of $F(x) = \chi_{\{f_L(\Theta,\cdot)>0.5\}}(x)$


Final result @ iteration = 20000

Intuition: billiards and geometrical optics

- Augmenting the dimension of the input data is similar to non-intersection phenomena in geometrical optics. Indeed, rays in a plane ($2d$) starting from different points can intersect, while this does not occur in a cylinder ($3d$).

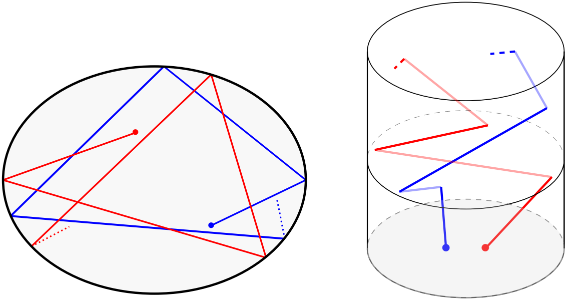


Figure: Rays may intercept in $2d$ (left) but not in $3d$ (right).

Universal approximation theorem(s)

An introduction

- Theorems generally of the form: *The class of neural networks is dense with respect to some topology in some function class $\mathcal{C}$.*
- **Interpretation**: given $f \in \mathcal{C}$ and $\varepsilon > 0$, there exists a neural network $f_{L(\varepsilon)}$ with $L(\varepsilon) > 0$ hidden layers, each layer consisting of $N_k(\varepsilon)$ components, as well as parameters $\Theta(\varepsilon)$, in a way that $\|f - f_\Theta((\varepsilon))\|_\mathcal{C} < \varepsilon$;
- Sometimes, $L > 0$ can be fixed too (see Slide 42). Then, more and more components in the hidden layers are needed to approximate f;
- The theorem does not say what these parameters are, or how to find them. It can basically be seen as an existence result for neural networks.

Theorem (Cybenko '89, MCSS)

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a nonconstant, bounded and continuous function.
Let $d \geq 1$ and $L = 1$.
For any $\varepsilon > 0$ and any $f \in C^0([0,1]^d)$, there exists $N_1 \in \mathbb{N}$, coefficients $A^0 \in \mathbb{R}^{N_1 \times d}$, $A^1 \in \mathbb{R}^{1 \times N_1}$ and $b^0 \in \mathbb{R}^{N_1}$ such that

$$f_L(\vec{x}) = A^1 \sigma(A^0 \vec{x} + b^0)$$

satisfies

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

- This theorem corresponds to $L = 1$ (one hidden layer) and $\varphi(x) = x$ (regression) in Definition Slide 14-15.
- Ingredients of the **proof**: Contradiction argument + Hahn Banach + Riesz-Representation + properties of σ.
- $\sigma(x) = \frac{1}{1+e^{-x}}$ fits the assumptions for instance.

We actually also have the following improved result.

Theorem (Poggio et al. '17)

Let $\sigma \in C^\infty(\mathbb{R})$ be bounded and not a polynomial.
Let $d \geq 1$ and $L = 1$.
For any $\varepsilon > 0$ and any $f \in C^\kappa([0,1]^d)$ with $\kappa \geq 1$ there exist coefficients $A^0 \in \mathbb{R}^{N_1 \times d}$, $A^1 \in \mathbb{R}^{1 \times N_1}$ and $b^0 \in \mathbb{R}^{N_1}$ with

$$N_1 = \mathcal{O}\left(\varepsilon^{-d/\kappa}\right)$$

such that

$$f_L(\vec{x}) = A^1 \sigma(A^0 \vec{x} + b^0)$$

satisfies

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

- Theorem implies that one can use a network with one hidden layer for approximating C^κ functions. However, in general the number N_1 of neurons needed for a fixed approximation accuracy $\varepsilon > 0$ grows exponen- tially in d, and so does the number of parameters in A^0, A^1, b^0. This means that the storage requirement as well as the effort to determine A^0, A^1, b^0 easily exeeds all reasonable bounds already for moderate dimensions d. Hence, this approach suffers from the curse of dimensionality.

More recent results for $\sigma(x) = \max(x, 0)$ 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 neural network f_L with ReLU activation of depth

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

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

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

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.
- Results for other function classes $(L^p, W^{k,p})$ exist as well.

Empirical guidelines

- It is true that any (say continuous) function can be approximated arbitrarily close both by a neural network and a polynomial. True for a lot of constructs (Galerkin, etc.).
- In machine learning one doesn't really want a function that fits through the data perfectly (overfitting). Rather, one wants something that fits well, but also probably works for points that have not been seen yet.
- Neural networks are (empirically) best-in-breed at solving a very specific kind of problem: computing a function $f : X \to Y$ given the values of f on a large but finite subset $A \subset X$. Typically Y (the space of "labels") is finite and small relative to A.
- As it turns out, neural networks can "learn" f so well that they can produce points in $f^{-1}(y)$'s near a prescribed $x \in X$, which is why they are good at image / language generation[4].

[4] Related to GAN (Generative adversarial networks), see Goodfellow et al. '16

Training

Training (Optimization of parameters)

- We are **given data** $\{(\vec{x}_i, \vec{y}_i)\}_{j=1}^{N} \in (\mathbb{R}^d)^N \times (\mathbb{R}^m)^N$;
- Training consists in **solving the optimization problem**:

$$\min_{\Theta=\{(A^k,b^k)\}_{k=0}^{L}} \sum_{i=1}^{N} |\vec{y}_i - f_\Theta(;\vec{x}_i)|^2 + \epsilon\mathcal{R}(\Theta); \tag{6}$$

$\epsilon > 0$ is a penalization/regularisation 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:

- Minimizer $\widehat{\Theta}$; if $\{\vec{y}_i\}_{i=1}^{N} \in \{0,1\}^N$ (*classification*), we set

$$F(\vec{x}) := \mathbb{1}_{\{f_L(\widehat{\Theta};\cdot)\geq\frac{1}{2}\}}(\vec{x}) \quad \forall \vec{x} \in \mathbb{R}^d$$

and we are done. Otherwise $F(\vec{x}) := f_L(\widehat{\Theta}, \vec{x})$ (*regression*).

Computing the minimizer

The functional to be minimized is of the form

$$J(\Theta) = \sum_{i=1}^{N} J_i(\Theta). \tag{7}$$

We could do gradient descent:

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

η is step-size. But often $N \gg 1$ ($N = 10^3$ or more).

- Stochastic gradient descent (Robbins-Monro 50s, Bottou et al, SIREV '18):
  1. pick $i \in \{1, \ldots, N\}$ uniformly at random
  2. $\Theta^{n+1} := \Theta^n - \eta \nabla J_i(\Theta^n)$
- *Mini-batch GD* can also be considered (pick a subset of data instead of just one point)
- Use chain rule and adjoints to compute these gradients ("backpropagation")
- Issues: might not converge to global minimizer; also how does one initialize the weights in the iteration (usually done at random)?

A discretized dynamical system

Recall that for a set of parameters $\Theta = \{(A^k, b^k)\}_{k=0}^{L}$ the neural network writes as

$$f_\Theta(;\vec{x}) := \varphi(\Lambda_{L+1} \circ \sigma \circ \Lambda_L \circ \sigma \ldots \circ \sigma \circ \Lambda_1)(\vec{x}) = \varphi(A^L z^L + b^L)$$

where for $\vec{x} \in \mathbb{R}^d$,

$$\begin{cases} z^{k+1} & = \sigma(A^k z^k + b^k) \quad \text{for } k = 0, \ldots, L-1 \\ z^0 & = \vec{x} \end{cases} \tag{8}$$

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

  $$\min_{\Theta = \{(A^k, b^k)\}_{k=0}^{L}} \sum_{i=1}^{N} |\vec{y}_i - \varphi(A^L z^L + b^L)|^2 + \epsilon \mathcal{R}(\Theta)$$

  with $z^L = z^L(\Theta, \vec{x}_i)$, subject to (**??**) with $z^0 = \vec{x}_i$.
- 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. Let $L \geq 1$ (at least 1 hidden layer).
A different *architecture* (Weinan E et al. '18): given any datum $\vec{x}_i \in \mathbb{R}^d$ for some $i = 1, \ldots, N$,

$$(9) \qquad \begin{cases} z^{k+1} & = z^k + \Delta t \, \sigma(A^k z^k + b^k) \quad \text{for } k = 0, \ldots L-1 \\ z^0 & = \vec{x}_i \in \mathbb{R}^d, \end{cases}$$

with $\Delta t = \frac{T}{L-1}$ and $T > 0$ given time horizon.

- requires same widths N_k at each layer $k \in \{0, \ldots, L\}$, but we can extend the data by 0.
- Recognize explicit Euler scheme for ODE

$$(10) \qquad \begin{cases} z'(t) = \sigma(A(t)z(t) + b(t)) \quad \text{for } t \in (0, T) \\ z(0) = \vec{x}_i \in \mathbb{R}^d. \end{cases}$$

- $z(t) = z_i(t)$, but A, b should be independent of the data.
- Universal approximation theorems exist also for ResNet architectures (see Lin & Jegelka '18)

The continuous Optimal Control Problem

- Can be advantageous to consider the continuous-time optimal control problem:

(11) $$\inf_{u(t)\in U,\,(\alpha,\beta)} \sum_{i=1}^{N} |\vec{y}_i - \varphi(\alpha\, z(T) + \beta)|^2 + \epsilon \mathcal{R}(u)$$

where $z = z_i$ solves

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

- Recall that often $\varphi \equiv \sigma$ (classification) or $\varphi(x) = x$ (regression).
- $u(t) = (A(t), b(t)) \in (\mathbb{R}^{d\times d} \times \mathbb{R}^d)^2$ and $F(u(t), z(t)) = \sigma(A(t)z(t) + b(t))$.
- $\sigma \in \mathrm{Lip}(\mathbb{R})$; existence of a minimizer see Trélat '05; here $U \subset L^\infty(0,T;(\mathbb{R}^{d\times d} \times \mathbb{R}^d)^2)$
- Easier to write optimality system (Weinan E et al. '18); we can use different algorithms for training (shooting method);
- Other schemes (Runge-Kutta) for time-discretization to obtain new architectures

Visualisation

Figure: We see that the trajectories of the dynamical system $z_i'(t) = \sigma(A(t)z_i(t) + b(t))$ with $z_i(0) = [x_i, 0]$ separate the points x_i.

Visualisation

Figure: We see that the trajectories of the dynamical system $z_i'(t) = \sigma(A(t)z_i(t) + b(t))$ with $z_i(0) = [x_i, 0]$ separate the points x_i. Here we ran the optimisation simulation on the same dataset.

Visualising the test error

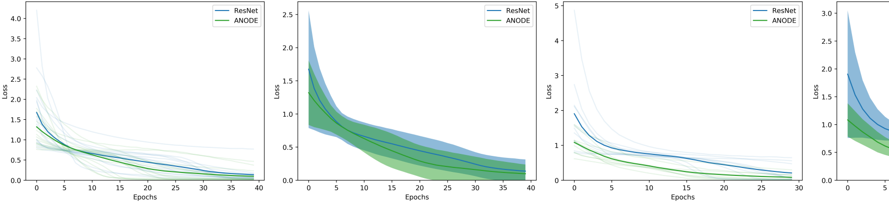


Figure: We ran the simulation for solving the optimisation problem 10 times on the same dataset as previous slide. The left figures are the losses in each run, with the mean shown in bold. On the right, we see the mean in bold, and the deviation is shaded. In blue, we have the ResNet neural network, while in green we solve the continuous ODEs with Runge-Kutta.

Discrete-time dynamical systems: A trick

Recall again that for parameters $\Theta = \{(A^k, b^k)\}_{k=0}^{L}$ the neural network writes[5]

$$f_\Theta(;\vec{x}) := \varphi(\Lambda_{L+1} \circ \sigma \circ \Lambda_L \circ \sigma \ldots \circ \sigma \circ \Lambda_2 \circ \sigma \circ \Lambda_1)(\vec{x}).$$

We can also (in addition to Slide 49) write $f_\Theta(;\vec{x}) = \varphi(z^L)$, where now

$$\begin{cases} z^{k+1} = A^{k+1}\sigma(z^k) + b^{k+1} & \text{for } k = 0, \ldots, L-1 \\ z^0 = A^0\vec{x} + b^0 \end{cases} \tag{13}$$

When $L \gg 1$, WLOG[6] assume $b^0 = 0$ and $A^0 = E : \mathbb{R}^d \to \mathbb{R}^{N_1}$ fixed embedding map. Then relabel the parameters Θ by shifting the indexes $k \leftarrow k+1$ and write

$$\begin{cases} z^{k+1} = A^k\sigma(z^k) + b^k & \text{for } k = 0, \ldots, L-1 \\ z^0 = E\vec{x}. \end{cases} \tag{14}$$

[5] Recall that $\Lambda_{k+1} z = A^k z + b^k \in \mathbb{R}^{N_{k+1}}$ for $z \in \mathbb{R}^{N_k}$ and $k \in \{0, \ldots, L\}$.

[6] If $L \gg 1$, we can choose to optimize the parameters A^k at a later hidden layer, not necessarily at the first one.

ResNets: Reformulation

Let $\{(\vec{x}_i, \vec{y}_i)\}_{i=1}^N \in (\mathbb{R}^d)^N \times (\mathbb{R}^m)^N$ be given training data. Let $m > 1$.
Now assume that $N_k = m$ for all $k \in \{1, \ldots, L+1\}$. We can consider

$$\text{(15)} \qquad \begin{cases} z^{k+1} = z^k + \Delta t \left(A^k \sigma(z^k) + b^k \right) & \text{for } k = 0, \ldots, L-1 \\ z^0 = E\vec{x}_i \in \mathbb{R}^m \end{cases}$$

for all i, where $\Delta t = \frac{T}{L-1}$ with $T > 0$ fixed.

- Can be seen as discretization of ODE: $z = z_i(t)$ solves

$$\text{(16)} \qquad \begin{cases} z'(t) = A(t)\,\sigma(z(t)) + b(t) & \text{in } (0,T) \\ z(0) = E\vec{x}_i \end{cases}$$

  for all $i \in \{1, \ldots, N\}$. Here $A(t) \in \mathbb{R}^{m \times m}$ and $b(t) \in \mathbb{R}^m$ are the controls.
- Compared to (**??**), (**??**) is an affine-control system (perhaps easier for theoretical purposes).
- **"Controllability"**: for $N \in \mathbb{N}$, find $A(t), b(t)$ such that $\varphi(z_i(T)) = \vec{y}_i$ for all $i \in \{1, \ldots, N\}$, where z_i solves (**??**).
- If $\varphi(x) = x$, we can try to apply Chow-Rashevski theorem (Iterated Lie brackets).

ResNets: Reformulation

Associated ResNet: assume that $N_k = m$ for all $k \in \{1, \dots, L+1\}$. We can consider

$$(17) \qquad \begin{cases} z^{k+1} = z^k + \Delta t \left(A^k \sigma(z^k) + b^k \right) & \text{for } k = 0, \dots, L-1 \\ z^0 = E\vec{x}_i \in \mathbb{R}^m \end{cases}$$

for all i, where $\Delta t = \frac{T}{L-1}$ with $T > 0$ fixed.
Now

- Can be seen as discretization of ODE: $z = z_i(t)$ solves

$$\begin{cases} z'(t) = A(t)\,\sigma(z(t)) + b(t) & \text{in } (0, T) \\ z(0) = E\vec{x}_i \end{cases}$$

for all $i \in \{1, \dots, N\}$. Here $A(t) \in \mathbb{R}^{m \times m}$ and $b(t) \in \mathbb{R}^m$ are the controls.

ResNets: Controllability

- Cuchiero, Larsson, Teichman '19 consider a simplified system

$$\begin{cases} z'(t) = \sum_{j=1}^{5} u^j(t)\, v^j(z(t)) & \text{in } (0,T) \\ z(0) = E\vec{x}_i \in \Omega \subset \mathbb{R}^m \end{cases} \tag{18}$$

- $v^j(\cdot) \in C^\infty(\mathbb{R}^m, \mathbb{R}^m)$ are polynomial fields at most of order 2; controls $u^j(t)$ are scalars.
- They prove Objective for (**??**) using (Ch-Ra) for stacked system satisfied by $\{z_i\}_{i=1}^N$

Theorem (Chow-Rashevski)

Let $\Omega \subset \mathbb{R}^m$ be bounded domain, and assume $v^1, \ldots, v^5$ satisfy the Hörmander condition

$$\textit{Lie}(v^1, \ldots, v^5)(x) = \mathbb{R}^m$$

*for all $x \in \Omega$. Then, (**??**) is exactly controllable (in usual sense).*

- Lie algebra evaluated at x:

  Lie$(v^1, \ldots, v^5)(x) := \{W(x) \colon W \in \text{span}\{v^1, \ldots v^5 \text{ and all iterated Lie brackets}\}\}$

- Lie bracket: $[u,v](x) = v'u - u'v$ for $u, v \in C^\infty(\mathbb{R}^m, \mathbb{R}^m)$ (v' and u' Jacobians).

How to quantify generalization?

- Let training data $\{\vec{x}_i, \vec{y}_i\}_{i=1}^N$ be samples of some unknown probability distribution p; f_Θ trained neural net
- Empirical risk (training error): $\mathcal{R}_N[f_\Theta] = \frac{1}{N}\sum_{i=1}^N \ell(f_\Theta(\vec{x}_i), \vec{y}_i)$
- Expected risk (non-sampled error):

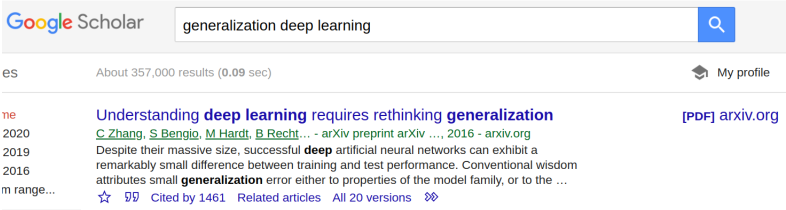
$$\mathcal{R}[f_\Theta] = \mathbb{E}_{x,y\sim p}[\ell(f_\Theta(x), y)]$$

(non-computable since p unknown)

- **Goal: minimize expected risk**. Possible approach: **bound**

$$\text{generalization gap} := \mathcal{R}_N[f_\Theta] - \mathcal{R}[f_\Theta]$$

by $\mathcal{O}((L/N)^\alpha)$, $\alpha > 0$, L is number of hidden layers.



generalization deep learning

About 357,000 results (**0.09** sec)

Understanding **deep learning** requires rethinking **generalization** [PDF] arxiv.org

C Zhang, S Bengio, M Hardt, B Recht… - arXiv preprint arXiv …, 2016 - arxiv.org

Despite their massive size, successful **deep** artificial neural networks can exhibit a remarkably small difference between training and test performance. Conventional wisdom attributes small **generalization** error either to properties of the model family, or to the …

☆ 99 Cited by 1461 Related articles All 20 versions »

Generalization is not well understood for deep neural nets.

Questions and 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?
- How does one best choose the widths N_k of the neural network w.r.t. the data?
- What mathematical control results does one transfer to deep learning (e.g. Lie brackets)?
- Other architectures were not presented (e.g. Convolutional Neural Networks)