# A mathematical introduction to Neural Networks and Neural Ordinary Differential Equations

Argimiro Arratia
argimiro@cs.upc.edu
http://www.cs.upc.edu/~argimiro

CS, Universitat Politécnica de Catalunya



# II. Neural Networks are Universal Approximators

## Summary: Neural Network paradigm

- Forward evaluation (training)

$$\mathbf{F}(\boldsymbol{x}) = \psi(W^{[D+1]}\mathbf{a}^{[D]} + \mathbf{b}^{[D+1]})$$

with $z^{[\mu]} = W^{[\mu]}\mathbf{a}^{[\mu-1]} + \mathbf{b}^{[\mu]}$ and $\mathbf{a}^{[\mu]} = \varphi(\mathbf{z}^{[\mu]})$, $\mu = 1, \ldots D$.

- Measure of quality of approximation (Cost function)

$$Cost(\theta) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{C}(\boldsymbol{y}^{[i]} - \boldsymbol{F}(\boldsymbol{x}^{[i]}))$$

- Backward propagation to improve approximation. By gradient descent update through layers

$$\theta \to \theta - \eta \nabla Cost(\theta)$$

**Remark:** The functions in $Cost$ are known and differentiable.



## Neural Networks as Universal Approximators

**The Representation Theorem (Hornik et al., Cybenko, 1989-91)**

Feed-forward network with one hidden layer of large enough width and a "squashing" activation function can approximate any integrable function to any accuracy.[a]

[a] Hornik, Stinchcombe, White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 359-366

**Remark** (Bruno Després) Let $f \in C^1(\mathbb{R})$

$$\begin{aligned} f(x) &= \int_{-\infty}^{x} f'(y)dy = \int_R H(x-y)f'(y)dy \\ &\approx \sum_{j=-J}^{J} \phi\left(\frac{x}{\epsilon} - \frac{j\Delta x}{\epsilon}\right) f'(j\Delta x)\Delta x = \sum_{j=-J}^{J} \omega_j \phi(a_j x + b_j) \end{aligned}$$

where $H(x)$ is Heaviside and $\phi$ a sigmoid to approximate $H$. Notice that $a, b$ tend to infinity with precision.

## Universal Approximation theorems

Let $\mathcal{N}(\sigma)$ be class of neural networks with fixed activation func. $\sigma$. $\mathcal{F}$ Banach function space with norm $\|\cdot\|_{\mathcal{F}}$

**universal approximation property**

Conditions under which $\mathcal{N}(\sigma)$ is *dense* in $\mathcal{F}$ w.r.to the topology induced by $\|\cdot\|_{\mathcal{F}}$
i.e. conditions such that $\forall f \in \mathcal{F}$ and $\epsilon > 0$, $\exists g \in \mathcal{N}(\sigma)$ such that $\|f - g\|_{\mathcal{F}} < \varepsilon$.

Universal approximation theory for NNet divides into:

- study of shallow networks (1-hidden layer) with arbitrarily large width, and
- deep neural networks (DNNs) with bounded width and arbitrarily large depth.



## Shallow networks approximation theory

Cybenko et al. (1989), Kurt Hornik (1991), Allan Pinkus (1999), and others. Consider multi-layer perceptron (MLP): $n$ input neurons, one output neuron, one hidden layer with an arbitrarily large width $k$.
Let $\sigma : \mathbb{R} \to \mathbb{R}$ and:

$$\mathcal{SN}_n^k(\sigma) = \{\sum_{i=1}^{k} c_i\, \sigma(\boldsymbol{w}^i x + b_i) \mid c_i, b_i \in \mathbb{R}, \boldsymbol{w}^i \in \mathbb{R}^n\} \quad (1)$$

$$\mathcal{SN}_n(\sigma) = \bigcup_{k=1}^{\infty} \mathcal{SN}_n^k(\sigma) = span\{\sigma(\boldsymbol{w}x + b) \mid b \in \mathbb{R}, \boldsymbol{w} \in \mathbb{R}^n\} \quad (2)$$

**Theorem (Universal approximation theorem)**

*Let $\sigma \in \mathcal{C}(\mathbb{R})$. Then $\mathcal{SN}_n(\sigma)$ is dense in $\mathcal{C}(\mathbb{R}^n)$, in the topology of uniform convergence on compacta, if and only if $\sigma$ is not a polynomial.*
*($\mathcal{C}(A) = \{f : A \to \mathbb{R} \mid f$ is continuous $\}$)*

## Ejercicio: Completar la Demostración del Teorema AU

- **Utilizar:** El conjunto de funciones Ridge

$$\mathcal{R} = span\{g(a \cdot x) | g \in \mathcal{C}(\mathbb{R}), a \in \mathbb{R}^n\}$$

  es denso en $\mathcal{C}(\mathbb{R}^n)$ .
  Para reducir la demostración de $\mathbb{R}^n$ a $\mathbb{R}$.
- Prop. 1: Si $\mathcal{SN}_1(\sigma)$ es denso en $\mathcal{C}(\mathbb{R})$ entonces $\mathcal{SN}_n(\sigma)$ es denso en $\mathcal{C}(\mathbb{R}^n)$
- Prop. 2: Sea $\sigma \in \mathcal{C}^\infty(\mathbb{R})$ y no un polinomio, entonces $\mathcal{SN}_1(\sigma)$ es denso en $\mathcal{C}(\mathbb{R})$.
  (Ayuda: usar Teorema Corominas-Sunyer (1954): Si $\sigma \in \mathcal{C}^\infty(\mathbb{R})$ en un intervalo abierto $A$ y no es un polinomio, entonces existe $b \in A$ tal que $\sigma^{(k)}(b) \neq 0$, $\forall k \geq 0$.
  Con el teorema anterior demostrar que $\mathcal{SN}_1(\sigma)$ contiene todos los monomios (y polinomios), y usar Teorema de Stone-Weierstrass.



## Arbitrary depth networks approximation theory

The Scenario: DNNs with bounded width and arbitrarily large number of layers.
Consider fully-connected DNN of input dimension $n$, $L$ hidden layers with $w$ neurons, and output dimension $m$:

$$\mathcal{DN}_w^L(\{\sigma_i\}) = \{\boldsymbol{W}^{[L+1]}\sigma(\boldsymbol{W}^{[L]}(\dots\sigma(\boldsymbol{W}^{[1]}x+\boldsymbol{b}^{[1]})\dots)+\boldsymbol{b}^{[L]})+\boldsymbol{b}^{[L+1]}\} \quad (3)$$

where at each layer we have activation function $\sigma \in \{\sigma_i\}$.
The family of arbitrarily deep DNN is:

$$\mathcal{DN}_w(\{\sigma_i\}) = \bigcup_{L=1}^{\infty} \mathcal{DN}_w^L(\{\sigma_i\}) \quad (4)$$

In this setting we have a critical threshold on the width $w_{min}$ of a neural network that allows it to be a universal approximator.

## Arbitrary depth NN

**Theorem (Yongqiang Cai, 2023)**

*For any compact domain $K \subset \mathbb{R}^n$ and any finite set of activation functions $\{\sigma_i\}$, $\mathcal{DN}_w(\{\sigma_i\})$ with width $w < w^*_{min} \equiv max\{n, m\}$ is not dense in $L^p(K, R^m)$ nor $\mathcal{C}(K, \mathbb{R}^m)$ in their respective usual topologies.*



## Arbitrary depth NN

This minimal width $w^*_{min}$ can indeed be reached.

**Theorem**

*Consider a compact $K \subset \mathbb{R}^n$ and*

$$ReLU(x) = max(0, x) \qquad FLOOR(x) = \lfloor x \rfloor$$

*Then, $\mathcal{DN}_w(\{FLOOR, ReLU\})$ with $w = max(n, m, 2)$, is dense in $\mathcal{C}(K, R^m)$ under the topology of uniform convergence on compacta.*

## Arbitrary depth NN

**Theorem**

*Consider a compact $K \subset \mathbb{R}^n$ and*

$$ABS(x) = |x|$$

$$\textit{leaky-ReLU}(x) = max(x, \alpha x) \quad \textit{for a fixed } \alpha \in [0, 1].$$

*Then, $\mathcal{DN}_{w^*_{min}}(\{ABS, \textit{leaky-ReLU}\})$ is dense in $L^p(K, R^m)$ in the usual topology.*



## References of NNet approximation theory

G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems 1989 2:4, 2:303-314, 12, 1989.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. Neural Networks, 4:251-257, 1, 1991.

Allan Pinkus. Approximation theory of the mlp model in neural networks. Acta Numerica, 8:143-195, 1999.

T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, Q. Liao, Why and when can deep but not shallow-networks avoid the curse of dimensionality: A review. Int. J. Autom. Comput. 14, 503-519 (2017).

T. Poggio, A. Banburski, Q. Liao. Theoretical issues in deep networks. PNAS v. 117 (2020)

Yongqiang Cai. Achieve the minimum width of neural networks for universal approximation. In The Eleventh International Conference on Learning Representations, 2023.

# Residual Neural Networks



## Motivation: Vanishing Gradient

Optimization of Nnet parameters is achieved by updating :

$$\theta \to \theta - \eta \nabla Cost(\theta)$$

layer through layer (gradient descent)
But it could happen that $\nabla Cost(\theta) \to 0$.

## Residual Neural Network (ResNet)

ResNet is a composition of *residual blocks*.
Let $\boldsymbol{h}_b \in \mathbb{R}^n$ be the hidden state before block $b \in \{0, ..., B\}$, and $\mathcal{F}(\cdot, \boldsymbol{\theta})$ a neural network with parameters $\boldsymbol{\theta}$. A residual block computes the next state by an additive transformation from the previous one:

$$\boldsymbol{h}_{b+1} = \boldsymbol{h}_b + \mathcal{F}(\boldsymbol{h_b}, \boldsymbol{\theta}) \tag{5}$$

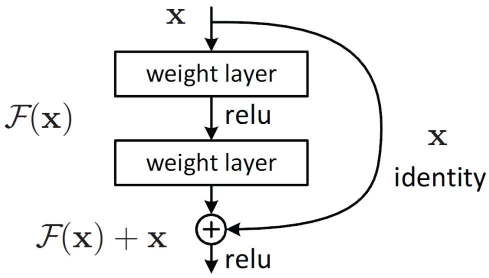



Figure: Sketch of a residual block



## ResNet: Forward propagation

If output of $l$-th residual block is input to the $(l+1)$-th residual block:

$$\boldsymbol{h}_{l+1} = \boldsymbol{h}_l + F(\boldsymbol{h}_l)$$

Apply the recursive formula, e.g.

$$\boldsymbol{h}_{l+2} = \boldsymbol{h}_{l+1} + F(\boldsymbol{h}_{l+1}) = \boldsymbol{h}_l + F(\boldsymbol{h}_l) + F(\boldsymbol{h}_{l+1})$$

we have

$$\boldsymbol{h}_L = \boldsymbol{h}_l + \sum_{i=l}^{L-1} F(\boldsymbol{h}_l)$$

where $L$ index of later (or last) block, $l$ index of earlier block. So there is always a signal directly sent from shallower block $l$ to deeper block $L$

## ResNet: Backward propagation

Given $Cost$ (or loss) func. to minimized, take derivative w.r.to $\boldsymbol{h}_l$:

$$\begin{aligned}\frac{\partial Cost}{\partial \boldsymbol{h}_l} &= \frac{\partial Cost}{\partial \boldsymbol{h}_L}\frac{\partial \boldsymbol{h}_L}{\partial \boldsymbol{h}_l} = \frac{\partial Cost}{\partial \boldsymbol{h}_L}\left(1 + \frac{\partial}{\partial \boldsymbol{h}_l}\sum_{i=1}^{L-1} F(\boldsymbol{h}_i)\right) \\ &= \frac{\partial Cost}{\partial \boldsymbol{h}_L} + \frac{\partial Cost}{\partial \boldsymbol{h}_L}\frac{\partial}{\partial \boldsymbol{h}_l}\sum_{i=1}^{L-1} F(\boldsymbol{h}_i)\end{aligned}$$

OBS. even if the gradients of $F(\boldsymbol{h}_i)$ terms are small, the total gradient $\frac{\partial Cost}{\partial \boldsymbol{h}_l}$ is NOT vanishing due to the added term $\frac{\partial Cost}{\partial \boldsymbol{h}_L}$



# Neural Ordinary Differential Equations (NODE)

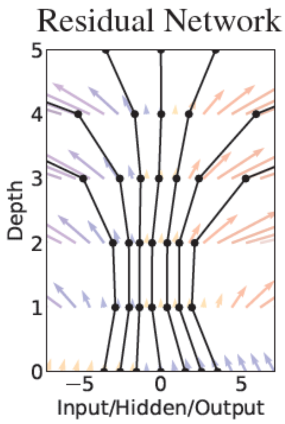
# From ResNet to NODE

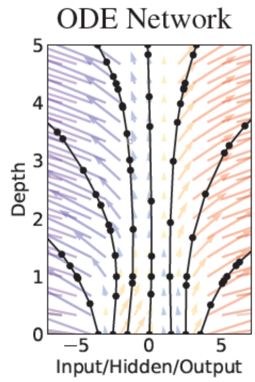
- Residual Network

$$h_{t+1} = h_t + f(h_t, \theta)$$

- Neural ODE[a]

$$\frac{h_{t+1} - h_t}{\Delta t} = \frac{f(h_t, \theta, t)}{\Delta t} \to \frac{dz}{dt} = f(z, \theta, t)$$




[a] Chen et al (2018) Neural ODE. In: Advances in Neural Information Processing Systems, 31



# Model as an IVP

- The model has become an Initial Value Problem.
  Let $z_0 := z(t_0) = \boldsymbol{x}$. Forward evaluation is

$$\boldsymbol{F}(z_0) = z(t_N) = z_0 + \int_{t_0}^{t_N} \frac{dz}{dt}\, dt = z_0 + \int_{t_0}^{t_N} f(z, \theta, t)\, dt$$

## Forward pass computes integration with ODE solver

For instance use Euler method to convert integral into many steps of addition

$$z(t+\epsilon) = z(t) + \epsilon \cdot f(z(t), \theta)$$

with $\epsilon < 1$.
Such ODE solvers are often numerically unstable (e.g. underflow error due to small step size, etc).
So, some other more sophisticated (black-box) ODE solvers are used.
**Remark.** $f(z(t), \theta)$, call it *the ODE function*, implicitly given from data, approximates $\frac{dz}{dt}$



## Optimization

- We can optimize: $\theta$, $t_0$, $t_N$ and $z_0$.
- Cost function

$$\text{Cost}\,(z(t_N)) = \text{Cost}\left(z(t_0) + \int_{t_0}^{t_N} f(z(t), \theta, t)\, dt\right)$$
$$= \text{Cost}\,(ODESolver(z(t_0), f, \theta, t_0, t_N))$$

- L1, L2, ...
- We need to calculate the following gradients

$$\frac{dCost}{dz(t_0)}, \frac{dCost}{d\theta}, \frac{dCost}{dt_0}, \frac{dCost}{dt_N}$$

## Adjoint sensitivity method I

As en example $\nabla_\theta Cost$. We want to find

$$\min_\theta Cost(z(t_N)) \quad \text{s.t.} \quad \frac{dz}{dt} = f(z, \theta, t)$$

Construct Lagrangian

$$\mathcal{L} = Cost(z(t_N)) - \int_{t_0}^{t_N} \lambda(t) \left( \frac{dz}{dt} - f(z, \theta, t) \right) dt$$

integration by parts and chain rule differentiation gives

$$\frac{dCost(z_{t_N})}{d\theta} = \int_{t_N}^{t_0} -a(t) \frac{\partial f}{\partial \theta} dt$$

with $a(t)$ the adjoint state, which is solution of IVP

$$a(t_N) = \frac{dCost(z_{t_N})}{dt_N}, \quad \frac{da}{dt} = -a(t) \frac{\partial f}{\partial z}$$

Further algebraic manipulation yields gradient of cost w.r.to $\theta$ is solution at time $t_0$ of IVP

$$a_\theta(t_N) = 0, \quad \frac{da_\theta}{dt} = -a(t) \frac{\partial f}{\partial \theta}$$



## Adjoint sensitivity method II

Similar calculations yield that the gradients of Cost w.r.to $z_{t_0}$, $t_0$ and $\theta$, all result from evaluating IVPs on corresponding adjoint states at time $t_0$.
Define augmented state $s(t) := [a(t), a_\theta(t), a_t(t)]$ as concatenation of adjoints for $z$, $\theta$ and $t$

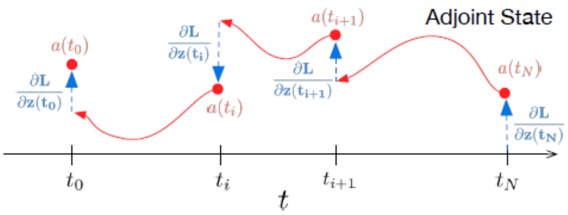
## Adjoint sensitivity method III

- Adjoint state at $t_0$

$$s(t_0) := \left[\frac{d\text{Cost}(z(t_N))}{dz(t_0)}, \frac{d\text{Cost}(z(t_N))}{d\theta}, -\frac{d\text{Cost}(z(t_N))}{dt_0}\right]$$

- Solving backwards Initial Value Problem

$$\begin{cases} s(t_N) = \left[\dfrac{d\text{Cost}(z_{t_N})}{dz_{t_N}}, \ \mathbf{0}, \ -a(t_N)f(z_{t_N}, \theta, t_N)\right] \\ \dfrac{ds(t)}{dt} = -a(t)\dfrac{\partial f}{\partial[z, \theta, t]} \end{cases}$$






## Neural ODE paradigm

### A Neural network with an ODE inside

- Forward evaluation: an Initial Value Problem

$$\boldsymbol{F}(z_0) = z(t_N) = z_0 + \int_{t_0}^{t_N} f(z, \theta, t)\, dt = ODESolver(z(t_0), f, \theta, t_0, t_N)$$

- Training (optimization): adjoint sensitivity method