



Workshop

Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

Taller

Coding Deep Neural Networks for PDEs

Session I: Introduction

Professors: Carlos Uriarte*, Ángel Javier Omella*, David Pardo*

Organizers: Ignacio Muga[†], Paulina Sepúlveda[†]

*Research Group on Applied Mathematical Modelling, Statistics, and Optimization (MATHMODE) at UPV/EHU

[†]IMA Numerics Research Group at PUCV

Follow us on: @mathmode.science*, @ima.numerics.pucv[†]

Biosfera Lodge, Olmué. January 19-24, 2025

Motivation: Partial Differential Equations

- Partial Differential Equations (PDEs) **model** multiple phenomena.

- For example:**

- Steady-state heat distribution in heterogeneous media via diffusion equation:

$$-\nabla \cdot \sigma \nabla u = f.$$

- Electromagnetic fields via Maxwell's equations (in the frequency domain):

$$\begin{cases} \nabla \times \mathbf{E} = -j\omega \mu \mathbf{H} - \mathbf{M}, & \text{Faraday's Law.} \\ \nabla \times \mathbf{H} = (\sigma + j\omega \epsilon) \mathbf{E} + \mathbf{J}, & \text{Amperè's Law.} \\ \nabla \cdot (\epsilon \mathbf{E}) = \rho_f, & \text{Gauss' Law of Electricity.} \\ \nabla \cdot (\mu \mathbf{H}) = 0, & \text{Gauss' Law of Magnetism.} \end{cases}$$

Parameters, Solution, Equation structure and source terms.

Motivation: Methods for solving linear PDEs

Numerical methods

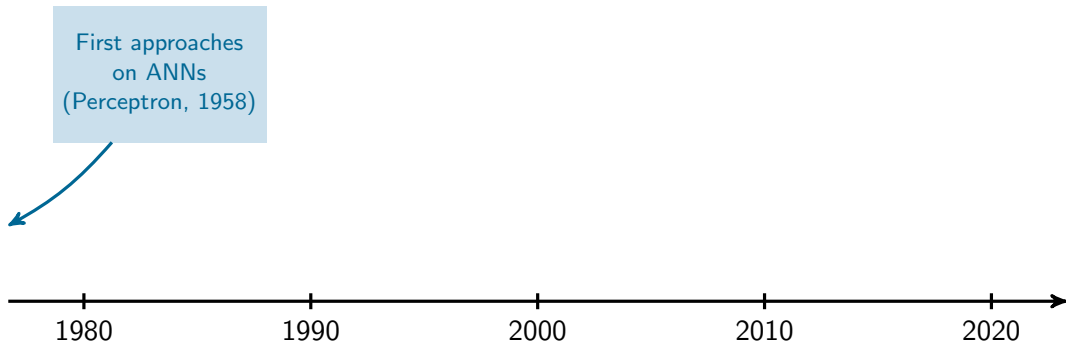
- Finite Difference Method (FDM)
- Finite Element Method (FEM)
- Spectral Methods (e.g., Fourier)

- Feed-Forward Neural Networks
- Convolutional Neural Networks
- Recurrent Neural Networks

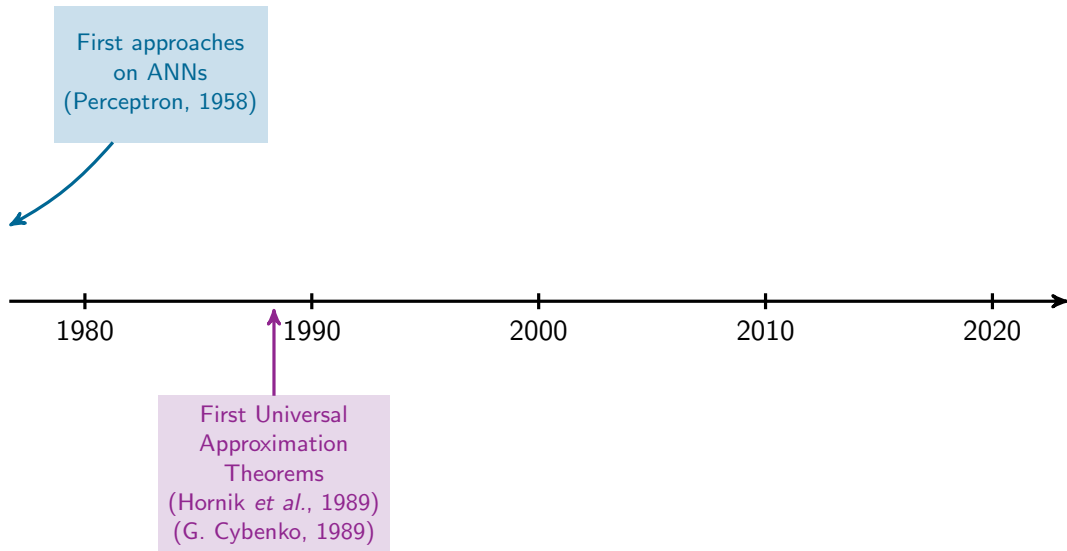
Advantages and limitations

- ✓ Converts the PDE into a system of linear equations (for a chosen finite basis)
 - ! Basis choice is critical
 - ✗ Curse of dimensionality
-
- ✓ Universal approximation property
 - ✓ Overcome the curse of dimensionality
 - ✗ Non-convex optimization
 - ✗ Integration is challenging

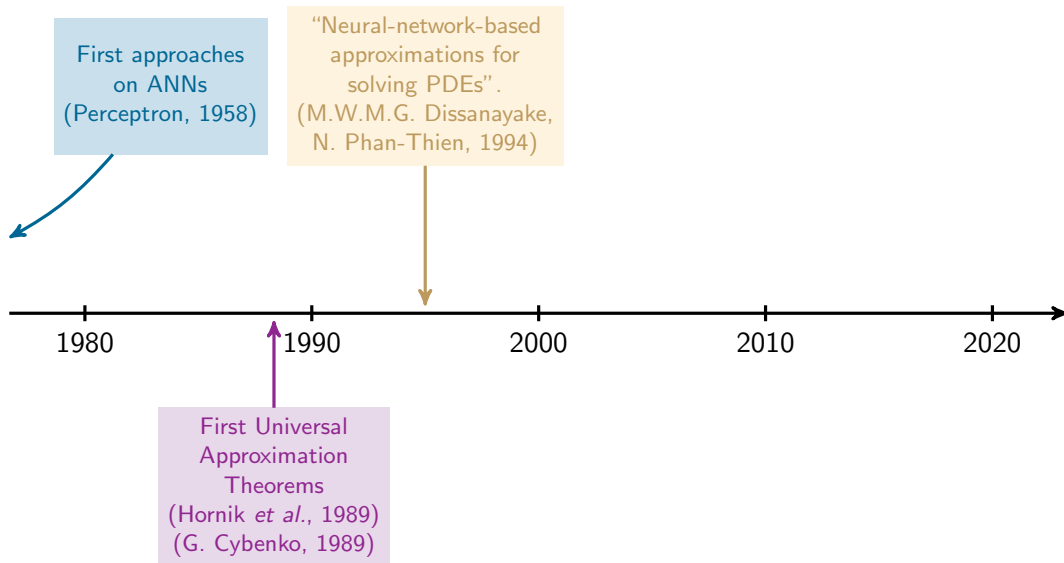
Timeline



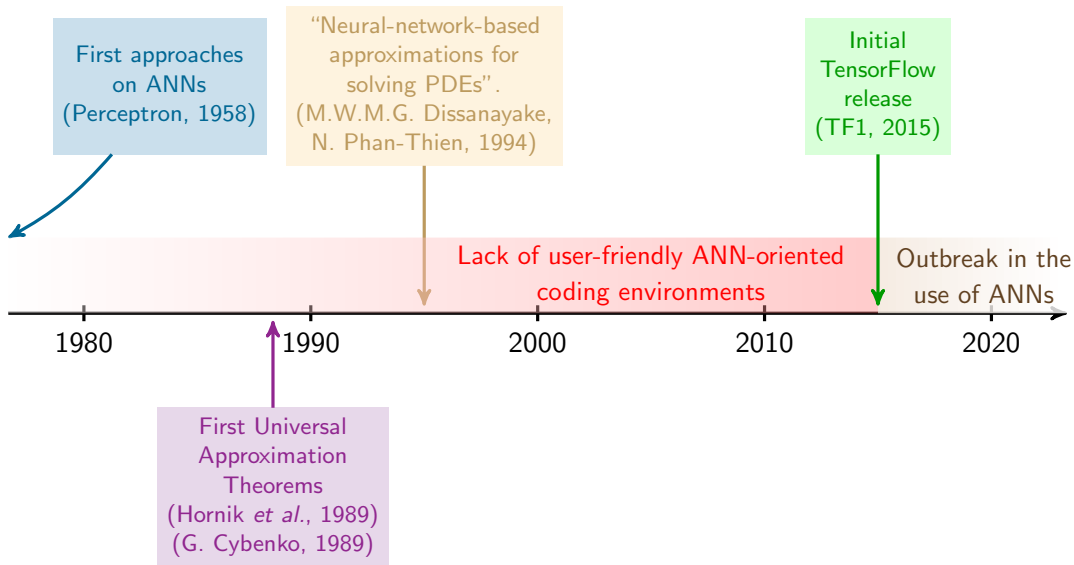
Timeline



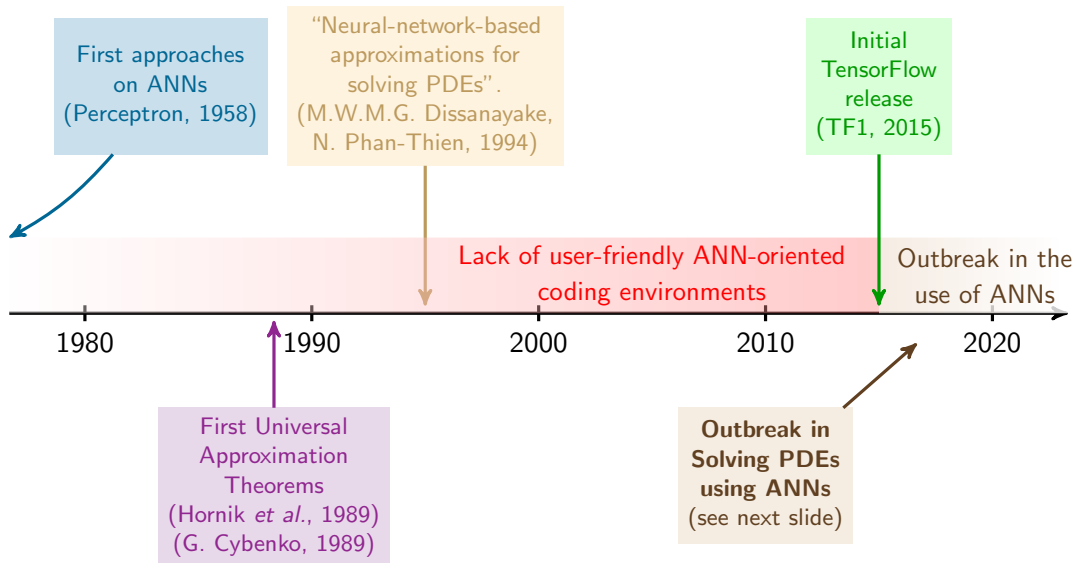
Timeline







Timeline



Timeline



Literature review

-  Yu, B. (2018). The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1), 1-12.
-  Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 686-707.
-  Kharazmi, E., Zhang, Z., & Karniadakis, G. E. (2021). hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374, 113547.
- ⋮
-  Uriarte, C. (2024). Solving Partial Differential Equations Using Artificial Neural Networks. *arXiv preprint arXiv:2403.09001*. PhD Dissertation (Section 1.4).
<https://arxiv.org/pdf/2403.09001>.

Minimization problem

- Assume that our problem of interest reformulates as

$$u^* := \arg \min_{u \in \mathbb{U}} \mathcal{L}(u), \quad \mathcal{L} : \text{loss/objective function}, \quad \mathbb{U} : \text{space of functions.}$$

E.g., To find u^* solving $Au^* = f$, we might consider $\mathcal{L}(u) = \|Au - f\|$.

- To approximate $u^* \in \mathbb{U}$, we proceed as follows:

1. Consider a **discretization** θ for \mathbb{U} , denoted $\theta \mapsto u(\theta) \in \mathbb{U}$.

2. “Seek” θ^* such that $\mathcal{L}(u(\theta^*)) = \inf_{\theta} \mathcal{L}(u(\theta))$. For simplicity, we will write:

$$\theta^* = \text{arg inf}_{\theta} \mathcal{L}(u(\theta)),$$

where “arg inf” should be read *loosely*.

By abuse of notation, θ denotes both the discretization mapping and the set of variables.

Discretization

Linear (Traditional) Approach

$$\theta \mapsto u(\theta)(x) := \sum_{j=1}^n \theta_j \psi_j(x),$$

$$\theta = (\theta_1, \theta_2, \dots, \theta_n)$$

where $\psi_j \in \mathbb{U}$ and $\theta_j \in \Theta = \mathbb{R}^n$.

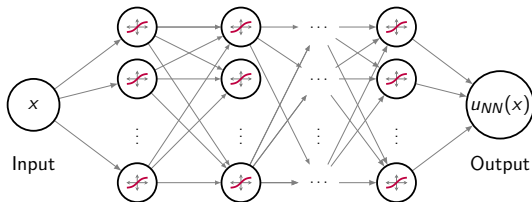
Typically:

- ψ_j is easy to implement (e.g., a polynomial).
- $\{\psi_j\}_{j=1}^n$ is linearly indep.

Artificial Neural Networks (ANNs)

$$\theta \mapsto u(\theta)(x) := u_{NN}(x),$$

$$\theta = \{\mathbf{W}_j, \mathbf{b}_j\}_{j=1}^k \cup \{\mathbf{W}\} \in \Theta = \mathbb{R}^n.$$

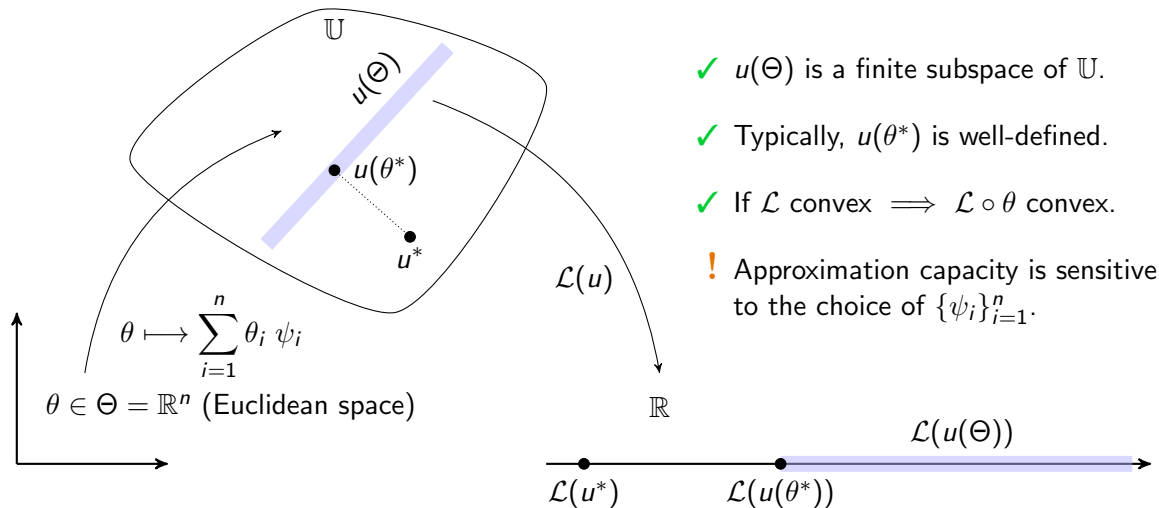


$$\mathbf{y}_0(x) := x \in \mathbb{R}^{n_0},$$

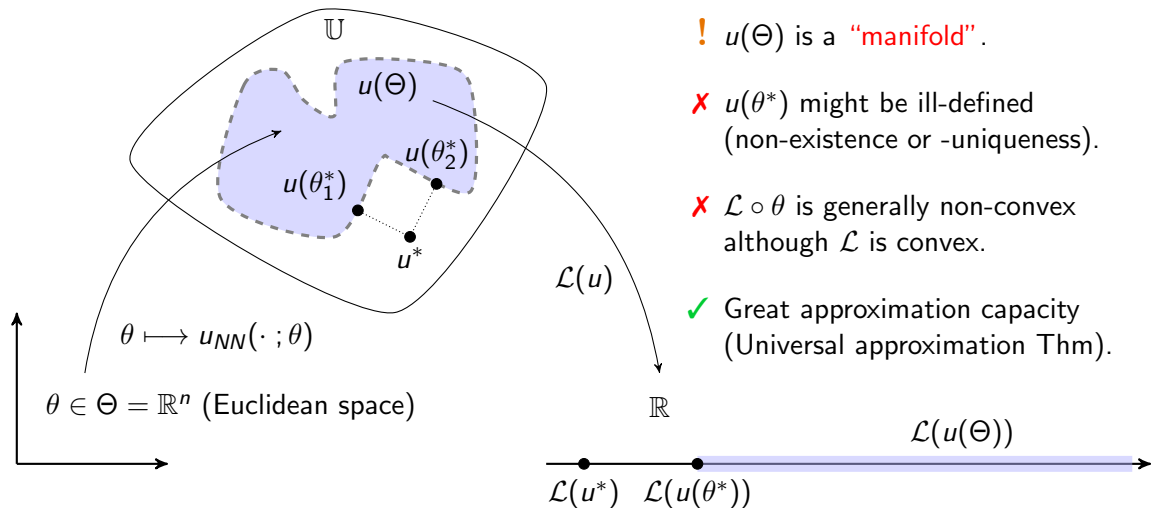
$$\mathbf{y}_j(x) := \varphi(\mathbf{W}_j \mathbf{y}_{j-1} + \mathbf{b}_j) \in \mathbb{R}^{n_j}, \quad 1 \leq j \leq k,$$

$$u_{NN}(x) := \mathbf{W} \mathbf{y}_k \in \mathbb{R}^{n_{k+1}},$$

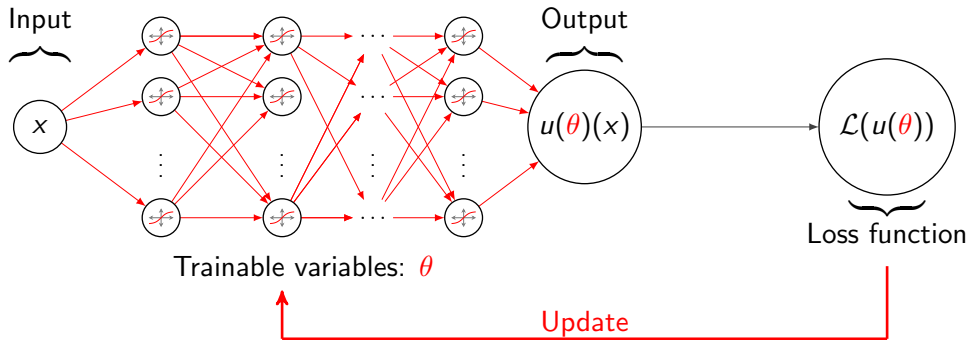
Linear-combination approach



ANN approach



Optimization



Optimization

- Typically performed in the form of a **first-order gradient-descent-based** methods.
- ✓ **Automatic differentiation** computes $\nabla_{\theta}\mathcal{L}(u(\theta))$ with a cost similar to evaluating $\mathcal{L}(u(\theta))$.
- ✗ Second-order methods (e.g., Newton methods) involve $\mathcal{H}_{\theta}(\mathcal{L}) := \nabla_{\theta}^2\mathcal{L}(u(\theta))$. Computing $\mathcal{H}_{\theta}(\mathcal{L})$ employing automatic differentiation is $\#\theta$ times more expensive than computing $\nabla_{\theta}\mathcal{L}(u(\theta))$. In ANN approaches, $\#\theta$ is huge!
- Continuum-level descriptions of gradient-descent and Newton optimization schemes,

$$\theta'_t = - \overbrace{\mathcal{H}_{\theta}(\mathcal{L})^{-1} \underbrace{\nabla_{\theta}\mathcal{L}(u(\theta_t))}_{\text{Gradient Desc.}}}^{\text{Newton step}}, \quad \begin{cases} \theta_t = \theta(t), \\ \theta_0 = \theta(0). \end{cases}$$

Optimization

- Iterative gradient-descent optimization:

$$\begin{cases} \theta'_t \approx \frac{\theta_{t+1} - \theta_t}{\eta} \\ \theta'_t = -\nabla_{\theta} \mathcal{L}(u(\theta_t)) \end{cases} \implies \theta_{t+1} = \theta_t - \eta \nabla_{\theta} \mathcal{L}(u(\theta_t))$$

- Adding momentum:

- At the continuum level:

$$\theta_t'' + \gamma \theta_t' = -\nabla_{\theta} \mathcal{L}(u(\theta_t)) \iff \begin{cases} \theta_t' &= v_t, \\ v_t' + \gamma v_t &= -\nabla_{\theta} \mathcal{L}(u(\theta_t)), \end{cases} .$$

- At the iterative level:

$$\begin{cases} v_{t+1} = \beta v_{t-1} - \eta \nabla_{\theta} \mathcal{L}(u(\theta_t)), \\ \theta_{t+1} = \theta_t + v_{t+1}, \end{cases} \quad 0 < \beta := 1 - \gamma\eta < 1.$$

- There exist multiple gradient-descent variants of this: with Nesterov acceleration, Adagrad, Adadelata, RMSprop, Adam/AdaMax, Nadam, etc.

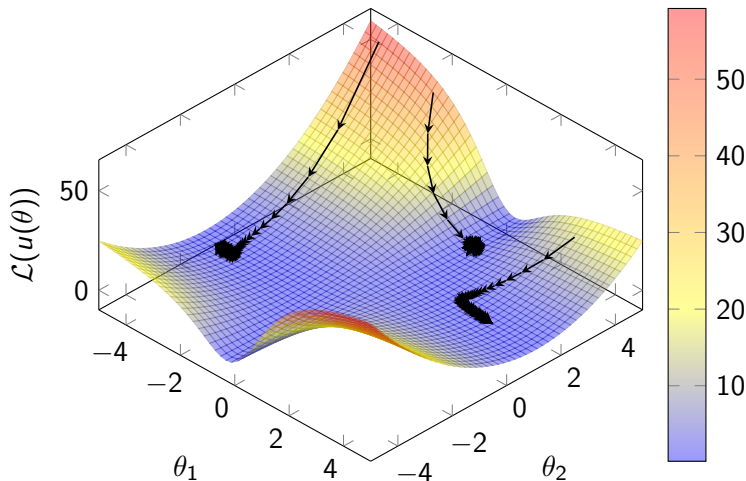
Optimization

Let $\theta = (\theta_1, \theta_2)$,

- $u(\theta)(x) = \theta_2 \tanh(\theta_1 x)$,
- $u^*(x) = \begin{cases} 1, & x > 0, \\ 0, & x = 0, \\ -1, & x < 0, \end{cases}$
- $\mathcal{L}(u) = \|u - u^*\|_{L^2(-1,1)}^2$.

Infimizer: $(\theta_1^*, \theta_2^*) = (\pm\infty, \pm 1)$.

We perform 200 gradient-descent iterations with $\eta = 0.05$.



Integration: General Comments

- **Exact integration** is typically **unavailable** due to the nonlinearity of activation functions.
- When our loss function has an integral form, we employ **numerical integration**:

$$\mathcal{L}(u(\theta)) = \int_{\Omega} \mathcal{I}_{u(\theta)}(x) dx \approx \sum_{k=1}^K \omega_k \mathcal{I}_{u(\theta)}(x_k).$$

- To avoid overfitting, **stochastic** quadrature is a must! (e.g., Monte Carlo integration)
- **Unbiasedness** of the chosen stochastic quadrature is another must! (e.g., MC is unbiased)

$$\mathbb{E} \left[\sum_{k=1}^K \omega_k \mathcal{I}(u(\theta))(X_k) \right] = \mathcal{L}(u(\theta)).$$

- Big **variance** entails big integration errors! (in MC, variance is of order $\mathcal{O}(\frac{1}{N})$).