Università degli Studi di Milano

MSc. in Data Science and Economics

Statistical Method for Machine Learning

Neural networks as universal approximators and an introduction to PINNs

Alessia Di Giovanni
alessia.digiovanni2@studenti.unimi.it

May 31, 2023

1 Problem statement and most important related works

Deep learning is a subset of machine learning that focuses on training neural networks with mutiple layers. They are inspired by the structure of human brain and have a large field of applications, such as image recognition,[8] natural language processing[5] and cognitive sciences.[10] This is due to the fact that neural networks have the capability of being universal approximators and for this reason they can also be used for solving ordinary and partial differential equations (ODEs and PDEs), with an approach called PINNs, physics-informed neural networks. This idea was introduced in the beginning of 1990s[11][13] but it was I. E. Lagaris with A. Likas and D. I. Fotiadis in 1998 who first had the intuition to use the function approximation capabilities of feedforward neural networks resulting in the construction of a solution written in differential, closed, analytic form, so a solution available in all the points of the domain and infinitely differentiable. In their paper *Artificial Neural Networks for Solving Ordinary and Partial Differential Equations*[9] they describe a method to solve single ordinary differential equations, systems of coupled ordinary differential equations and partial differential equations by means of constructing a trial solution and numerical optimization procedure.[14] But in their work the term PINNs is not used and a first definition of it can be found in *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations* by M.Raissia, P.Perdikarisb, G.E.Karniadakisa.[15] They define PINNs as *"neural networks that are trained to solve supervised learning tasks while respecting any given laws of physics described by general nonlinear partial differential equations"*. The need to construct a trial solution that satisfies the initial/boundary conditions associated with the differential equation is eliminated and they train the neural network itself for satisfying the initial/boundary conditions. In fact the construction of a trial solution could be an optimal method in case of differential equations with relatively simple initial/boundary conditions but it could become a challenging task if they are complex or with higher dimensional differential equations.

2 A little recap on differential equations[1]

Definition 2.1. A **differential equation** (DE) is a mathematical equation for an unknown function of one or several variables that relate the function to its derivatives.

Definition 2.2. A **solution** to a differential equation is a function that satisfies the differential equation.

Definition 2.3. The **order** of a differential equation is the order of the largest derivative of the unknown function that appears in the equation.

Boundary conditions in a differential equation are the specified values or constraints imposed on the function at certain points or intervals of the independent variables. If a solution satisfies all the specified boundary conditions, it is considered the unique solution to that particular equation. This principle is known as the **uniqueness theorem**, stating that a set of boundary conditions ensures the uniqueness of the solution. Examples of boundary conditions are Dirichlet boundary conditions, specifying the value that the unknown function needs to take on along the boundary of the domain, and Neumann boundary conditions, specifying the values that the derivative of a solution is going to take on the boundary of the domain. A second constraint that can be imposed to differential equations are the **initial conditions**, the value that the unknown function takes when the independent variable is at the staring point.

Definition 2.4. Based on the number of independent variables, there are two types of differential equations:

- **ordinary differential equations** (ODEs): differential equations where the derivatives are taken with respect to only one variable. That is, there is only one independent variable;
- **partial differential equations** (PDEs): differential equations that depend on partial derivatives of several variables. That is, there are several independent variables.

Definition 2.5. Based on how the dependent variable and its derivatives appear in the equation, differential equations can be

- **linear**: the dependent variable and its derivatives appear linearly, that is, only as first powers and not multiplied together;
- **non-linear**: the dependent variable and its derivatives appear nonlinearly. This means that the variables may be raised to powers other than one, multiplied together, or involve other nonlinear functions (for example exponential, trigonometric, logarithmic functions).

A famous example of second order non-linear partial differential equation is the Burger's equation, described in the section 4. Non-linear differential equations can have more complex behaviors compared to linear equations and they often arise in many scientific and engineering applications where non-linear phenomena are present. Examples include biological systems, chemical reactions, population dynamics, fluid flow, electrical circuits and many other physical processes. Solving non-linear differential equations analytically can be challenging and, in many cases, exact solutions may not be obtainable. Therefore, numerical methods and techniques are commonly employed to approximate the solutions of non-linear differential equations, such as the **Finite Element Method**. But FEM involves dividing the domain into a large number of small elements and nodes, and solving a system of equations for each node. This can result in a huge amount of data and calculations, especially for three-dimensional problems and fine meshes. This is why the introduction of neural networks to solve especially this more complex case of differential equations bringing a solution that is differentiable and in closed analytic form easily used in any subsequent calculation (other techniques offer discrete solution or solution of limited differentiability).

The paper is divided in two sections: in the first one the fundamental property of neural networks is shown; then it follows the description of the method to solve non-linear partial differential equations using PINNs.

3 Multilayered feedforward neural networks are universal approximators

When we deal with neural networks, it is obvious to introduce their capability of universal function approximators so they can also approximate the solution for differential equations. In this section it follows the mathematical demonstration of why feedforward neural networks with as few as one hidden layer are universal function approximators, following the paper *Multilayer feedforward networks are universal approximators* by Hornik.[6]

Definition 3.1. Given $r \in \mathbb{N}$, A^r is the set of all **affine** functions

$$\begin{aligned} f &: \mathbb{R}^r \to \mathbb{R} \\ f(\underline{x}) &= \underline{w} \cdot \underline{x} + b \end{aligned} \tag{1}$$

where $\underline{w}$ and $\underline{x}$ are vectors in $\mathbb{R}^r$ and $b \in \mathbb{R}$ is a scalar.

In this section, $\underline{x}$ is the network input, $\underline{w}$ corresponds to network weights from input to the intermediate layer and the scalar b corresponds to the bias.

Now recall the definitions of topological space, continuous function, σ-algebra, measurable space and measurable function.[16]

Definition 3.2. A collection τ of subsets of a set X is said to be a **topology** in X if τ satisfies the following four conditions:

- the empty set $\emptyset$ is in τ;
- X is in τ;

- the intersection of a finite number of sets in τ is also in τ;
- the union of an arbitrary number of sets in τ is also in τ.

Definition 3.3. If τ is a topology in X, then X is called **topological space** and the members of τ are called **open sets**.

Definition 3.4. If X and Y are topological spaces and G is a mapping of X into Y, then G is said to be **continuous** if

$$G^{-1}(V) \text{ is an open set in X}, \ \forall \text{ V open set in Y}.$$

Definition 3.5. Given a set X, a family $\mathcal{M}$ of subsets in X is called σ**-algebra** if

- X is in $\mathcal{M}$;
- $B \in \mathcal{M} \Longrightarrow B^C \in \mathcal{M}$, where B^C is the complement of B relative to X;
- $B_n \in \mathcal{M}, n \geq 1 \Longrightarrow \bigcup_{n\geq 1} B_n \in \mathcal{M}$.

Definition 3.6. If $\mathcal{M}$ is a σ-algebra in X, the pair $(X, \mathcal{M})$ is called **measurable space** and the elements of $\mathcal{M}$ are called **measurable sets**.

Definition 3.7. Given $(X, \mathcal{M})$ a measurable space, Y a topological space and G a mapping of X into Y, then G is said to be **measurable** if

$$G^{-1}(V) \in \mathcal{M}, \ \forall \, V \text{ open set in Y}.$$

Definition 3.8. Given $G(\cdot)$ any measurable function mapping $\mathbb{R}$ to $\mathbb{R}$ and $r \in \mathbb{N}$, $\Sigma^r(G)$ is the class of functions

$$\left\{ h : \mathbb{R}^r \to \mathbb{R} : h(\underline{x}) = \sum_{j=1}^{q} \beta_j G(A_j(\underline{x})), \quad \underline{x} \in \mathbb{R}^r, \quad \beta_j \in \mathbb{R}, \quad A_j \in A^r, \quad q = 1, 2, ... \right\} \tag{2}$$

If G is taken as a squashing function, $\Sigma^r(G)$ is the class of output functions for a single hidden layer feedforward networks. The input to the neural network is the r-dimensional vector $\underline{x}$, each term in the sum corresponds to a separate hidden unit in the hidden layer where the scalar β_j are the weights from hidden to output layers. $A_j(\underline{x})$ represents the input to the hidden unit. The output of the neural network is the sum of the outputs from the hidden layer units.

Definition 3.9. A function $\Psi : \mathbb{R} \to [0, 1]$ is a **squashing function** if it is non-decreasing:

$$\begin{cases} lim_{\lambda \to \infty} \Psi(\lambda) = 1 \\ lim_{\lambda \to -\infty} \Psi(\lambda) = 0 \end{cases} \tag{3}$$

An example of squashing function is the **sigmoid function** defined as:

$$\sigma(\lambda) = \frac{1}{1 - e^{-\lambda}}. \tag{4}$$

Squashing functions have at most countable many discontinuities so they are measurable.

Definition 3.10. Given $G(\cdot)$ any measurable function mapping $\mathbb{R}$ to $\mathbb{R}$ and $r \in \mathbb{N}$, $\Sigma\Pi^r(G)$ is the class of function

$$\left\{ h : \mathbb{R}^r \to \mathbb{R} : h(\underline{x}) = \sum_{j=1}^{q} \beta_j \prod_{k=1}^{l_j} G(A_{jk}(\underline{x})), \quad \underline{x} \in \mathbb{R}^r, \quad \beta_j \in \mathbb{R}, \quad A_{jk} \in A^r, \quad l_j \in N, \quad q = 1, 2, ... \right\} \tag{5}$$

Note that if $l_j = 1$ for all j, the definition 3.10 is the definition 3.8.

Theorem 3.1. *If $\mathcal{F}$ is a collection of subsets of X, there exists a smallest σ-algebra $\mathcal{M}^*$ in X such that $F \subset \mathcal{M}^*$.*

Proof. Let Ω be the family of all σ-algebras $\mathcal{M}$ in X which contains $\mathcal{F}$. Since the collection of all subsets of X is such a σ-algebra, Ω is not empty. Let $\mathcal{M}^*$ be the intersection of all $\mathcal{M} \in \Omega$. It follows

- $\mathcal{F} \subset \mathcal{M}^*$;
- $\mathcal{M}^*$ is in every σ-algebra in X wich contains $\mathcal{F}$.

It remains to be shown that $\mathcal{M}^*$ is a σ-algebra.

- X is in every $\mathcal{M} \in \Omega$ because $\mathcal{M}$ is a σ-algebra, so it belongs to the intersection, $X \in \mathcal{M}^*$;
- if B is in $\mathcal{M}^*$, it is in every $\mathcal{M} \in \Omega$. Since $\mathcal{M} \in \Omega$ are σ-algebra, $B^C \in \mathcal{M} \;\; \forall \mathcal{M} \in \Omega$, so B^C is in the intersection of all $\mathcal{M}$, $B^C \in \mathcal{M}^*$;
- if $A_n \in \mathcal{M}^*$, n=1,2,3..., $A_n \in \mathcal{M}$ and $\bigcup A_n \in \mathcal{M}$, because $\mathcal{M}$ is a σ-algebra. Since $\bigcup A_n \in \mathcal{M}$ for every $\mathcal{M} \in \Omega$, $\bigcup A_n$ lies also in the intersection, $\bigcup A_n \in \mathcal{M}^*$.

□

Definition 3.11. Let X be a topological space. By the previous theorem, there exists a smallest σ-algebra $\mathcal{B}$ in X such that every open set in X belongs to $\mathcal{B}$. The members of $\mathcal{B}$ are called **Borel sets**.

Since $\mathcal{B}$ is a σ-algebra, we may now regard X as a measurable space, with the Borel sets playing the role of the measurable sets; more concisely, we consider the measurable space $(X, \mathcal{B})$. If $G : X \to Y$ is a continuous mapping of X, where Y is any topological space, then it is evident from the definitions that $G^{-1}(V) \in \mathcal{B}$ for every open set V in Y. In other words, every continuous mapping of X is Borel measurable. Borel measurable mappings are often called Borel mappings, or Borel functions.

Let C^r the set of continuous functions from $\mathbb{R}^r$ to $\mathbb{R}$ and M^r the set of all Borel measurable functions from $\mathbb{R}^r$ to $\mathbb{R}$: for the statement above C^r is a subset of M^r. The result concering the approximation of functions is first demonstrated for functions in C^r and then extended for functions in M^r.

Definition 3.12. A **metric space** is a set X in which a **distance function** (or **metric**) ρ is defined, with the following properties:

- $0 \le \rho(x, y) \le \infty$ for all x and y in X;
- $\rho(x, y) = 0$ if and only if $x = y$;
- $\rho(x, y) = \rho(y, x)$ for all x and y in X;
- $\rho(x, y) \le \rho(x, z) + \rho(z, y)$ for all x, y and $z \in X$ (*triangle inequality*).

Definition 3.13. A subset S of a metric space (X, ρ) is **ρ-dense** in a subset T if for every $\epsilon > 0$ and for every $t \in T$, there is a $s \in S$ such that $\rho(s, t) < \epsilon$.

So an element of S can approximate an element of T with any degree of accuracy. In our case, S corresponds to $\Sigma\Pi^r(G)$ (or $\Sigma^r(G)$) and T corresponds to M^r (or C^r). As said above, let's start considering C^r.

Definition 3.14. Let Y be a topological space. A subset $K \subset Y$ is said to be **compact** if for any open cover $\{U_i\}_{i \in I}$ of K, there exists a finite subcover $\{U_{i1}, U_{i2}, ..., U_{in}\}$ that covers K.

In other words, for any collection of open sets that covers K, it is possible to select a finite number of open sets from that collection that still covers K.

Definition 3.15. A subset S of C^r is **uniformly dense** on compacta in C^r if for every compact subset $K \subset \mathbb{R}^r$, S is ρ_k-dense in C^r, where $\rho_k(f,g) = sup_{x \in K}|f(x) - g(x)|$, $f, g \in C^r$.

Theorem 3.2. *Let G be any continuous noncostant function from $\mathbb{R}$ to $\mathbb{R}$. Then $\Sigma\Pi^r(G)$ is uniformly dense on compacta in C^r.*

The theorem states that for any function in the set C^r, it is possible to find $\Sigma\Pi$ feedforward network that can approximate the function arbitrarily closely on any compact subset of its domain.

For the proof of the theorem, *Stone-Weierstrass theorem* has to be introduced. But first, some definitions must be given.

Definition 3.16. A family A of real functions defined on a set E is an **algebra** if A is closed under addition, multiplication and scalar multiplication.

Definition 3.17. A family A **separate points** on E if for every x, y in E, $x \neq y$, there exists a function f in A such that $f(x) \neq f(y)$.

Definition 3.18. A family A **vanishes at no point** of E if for each x in E there exists f in A such that $f(x) \neq 0$.

Theorem 3.3 (Stone-Weierstrass Theorem)**.** *Let A an algebra of real continuous functions on a compact set K. If A separates points on K and if A vanishes at no point of K, then the uniform closure B of A consists of all real continuous functions on K (i.e., A is ρ_k-dense in the space of real continuous functions on K).*

Now, the proof of theorem 3.2 can be given.

Proof. Let $K \in \mathbb{R}^r$ be any compact set. For any G, $\Sigma\Pi^r(G)$ is an algebra on K, because the closure under addition, under multiplication and under scalar multiplication are satisfied.

Let $h_1, h_2 \in \Sigma\Pi^r(G)$, which means

$$h_1(\underline{x}) = \sum_{i=1}^{q_1} \beta_j^{(1)} \prod_{k=1}^{l_j^{(1)}} G(A_{jk}^{(1)}(\underline{x})) \qquad and \qquad h_2(\underline{x}) = \sum_{i=1}^{q_2} \beta_j^{(2)} \prod_{k=1}^{l_j^{(2)}} G(A_{jk}^{(2)}(\underline{x})). \tag{6}$$

Now, consider $h(\underline{x}) = h_1(\underline{x}) + h_2(\underline{x})$. We can express $h(\underline{x})$ as:

$$h(\underline{x}) = \sum_{i=1}^{q_1} \beta_j^{(1)} \prod_{k=1}^{l_j^{(1)}} G(A_{jk}^{(1)}(\underline{x})) + \sum_{i=1}^{q_2} \beta_j^{(2)} \prod_{k=1}^{l_j^{(2)}} G(A_{jk}^{(2)}(\underline{x})) \tag{7}$$

Since addition and multiplication are commutative and associative operations, we can rearrange terms to obtain:

$$h(\underline{x}) = \sum_{i=1}^{q} \beta_j \prod_{k=1}^{l_j} G(A_{jk}(\underline{x})) \tag{8}$$

where $q = q_1 + q_2$, $\beta_j = \beta_j^{(1)}$ for $j = 1, 2, ..., q_1$, and $\beta_j = \beta_{j-q_1}^{(2)}$ for $j = q_1 + 1, q_1 + 2, ..., q$. Thus, $h(\underline{x})$ is a finite sum of products of functions in $G(\cdot)$, which demonstrates closure under addition.

Consider now $h(\underline{x}) = h_1(\underline{x}) \cdot h_2(\underline{x})$. Using distributivity and rearranging terms, we can express $h(\underline{x})$ as a finite sum of products of functions in $G(\cdot)$, which shows closure under multiplication.

In order to show closure under multiplication with a scalar, we need to demonstrate that for any $h \in \Sigma\Pi^r(G)$ and $\alpha \in \mathbb{R}$, the function αh belongs to $\Sigma\Pi^r(G)$. Finally let's define αh as follows:

$$(\alpha h)(\underline{x}) = \sum_{i=1}^{q} (\alpha\beta_j) \prod_{k=1}^{l_j} G(A_{jk}(\underline{x})) \tag{9}$$

where α is a scalar. By the closure of $\Sigma\Pi^r(G)$ under multiplication, we know that each term in the sum, $(\alpha\beta_j)\prod_{k=1}^{l_j} G(A_{jk}(\underline{x}))$, belongs to $\Sigma\Pi^r(G)$. Therefore, the function αh can be expressed as a finite sum of products of functions in $G(\cdot)$, and it belongs to $\Sigma\Pi^r(G)$.

Having shown closure under addition, multiplication and multiplication with a scalar, it can be concluded that $\Sigma\Pi^r(G)$ is an algebra on the compact set K in $\mathbb{R}^r$.

If $x, y \in K$, $x \neq y$, then there is an $A \in A^r$ such that $G(A(x)) \neq G(A(y))$. In fact, let $a, b \in \mathbb{R}$, $a \neq b$ such that $G(a) \neq G(b)$, A can be choosen such that $A(x) = a$ and $A(y) = b$, so $G(A(x))$ will be different from $G(A(y))$. $\Sigma\Pi^r(G)$ separates points on K.

Then let's show that $\Sigma\Pi^r(G)$ vanishes at no point in K. Pick $b \in \mathbb{R}$ such that $G(b) \neq 0$ and set $A(x) = 0 \cdot x + b$. For all $x \in K$, $G(A(x)) = G(b)$: this is the f that ensures that $\Sigma\Pi^r(G)$ vanishes at no point in K.

From the Stone-Weierstrass theorem it follows that $\Sigma\Pi^r(G)$ is ρ_k-dense in the space of real continuous functions on K. Because K is arbitrary, the result follows. □

Definition 3.19. Given a probability measure μ on $(\mathbb{R}^r, \mathcal{B}^r)$, the metric ρ_μ from M^r x M^r to $\mathbb{R}^+$ is defined as

$$\rho_\mu(f,g) = inf\{\epsilon > 0 : \mu\{x : |f(x) - g(x)| > \epsilon\} < \epsilon\} \tag{10}$$

In the following theorem, we pass from C^r to M^r.

Theorem 3.4. *For every continuous nonconstant function G, every r and every probability measure μ on $(\mathbb{R}^r, \mathcal{B}^r)$, $\Sigma\Pi^r(G)$ is ρ_μ-dense in M^r.*

The theorem states that within the space of measurable functions, it is possible to approximate any function using a function in $\Sigma\Pi^r(G)$, provided that G is continuous and non-constant.

For the proof of the theorem, a lemma is needed which relates uniform convergences on compacta to ρ_μ-convergence.

Definition 3.20. A sequence of functions $\{f_n\}$ is said to be **uniformly convergent** to a function f if for all compact $K \subset \mathbb{R}^r$ $\rho_K(f_n, f) \to 0$ as $n \to \infty$.

Lemma 3.5. *If $\{f_n\}$ is a sequence of functions in M^r that converges uniformly on compacta to the function f then $\rho_\mu(f_n, f) \to 0$.*

Lemma 3.6. *For any finite measure μ, C^r is ρ_μ-dense in M^r.*

The proof of the theorem 3.4 can be given.

Proof. Given G any continuous noncostant function, from theorem 3.2 $\Sigma\Pi^r$ is uniformly dense on compacta in C^r. From lemma 3.5 it follows that $\Sigma\Pi^r(G)$ is ρ_μ-dense in C^r. Because C^r is ρ_μ-dense in M^r by lemma 3.6, it follows that $\Sigma\Pi^r(G)$ is ρ_μ-dense in M^r applying the triangle inequality. To see why this follows, consider any function f in M^r. By the definition of ρ_μ-density, there exists a function g in C^r such that $\rho_\mu(f,g)$ is arbitrarily small ($< \frac{\epsilon}{2}$). Similarly, since $\Sigma\Pi^r(G)$ is ρ_μ-dense in C^r, there exists a function h in $\Sigma\Pi^r(G)$ such that $\rho_\mu(g,h)$ is arbitrarily small ($< \frac{\epsilon}{2}$). Using the triangle inequality

$$\rho_\mu(f,h) \leq \rho_\mu(f,g) + \rho_\mu(g,h) < \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon. \tag{11}$$

□

If G is taken as a squashing function, the continuity is not required. From the last theorem it follows that $\Sigma\Pi$ are universal approximators and for this reason neural networks can be used to solve differential equations.

4 Multilayered feedforward neural networks to solve non linear partial differential equations

The main reason why physics-informed neural networks are introduced is because data-driven models may fit observations very well but their predictions may lack physical consistency or plausibility. Therefore, there is a need to integrate fundamental physical laws, described by differential equations, by teaching machine learning models about the governing principles of physics. In this way an "informative prior" can be used to improve the performance of a learning algorithm, particularly in the presence of imperfect data. The optimal scenario arises when there is a balance between the quantity of available data, scattered measurements of primary or auxiliary states, which can be utilized to infer parameters and even missing terms in the PDE, while simultaneously recovering the solution, and the underlying physics.[7]

In this section it follows the description of the method used to solve a non-linear partial differential equations using physics-informed neural networks. Let $u(x,t)$ be an unknown function; a general second order non-linear partial differential equation is given by :

$$F(f(x,t), u(x,t), \nabla(u(x,t)), \nabla^2(u(x,t)) = 0 \tag{12}$$

where F is a non-linear function. The two independent variables of the equation are x and t. In order to understand better the procedure,[14] let's consider an example, the one-dimensional Burger's equation.[12]

The equation was first introduced by Harry Bateman in 1915[3] and later studied by Johannes Martinus Burgers in 1948.[4] It is used in fluid dynamics, particularly in the study of shock waves and non-linear wave propagation. It is given by

$$u_t + uu_x = \nu u_{xx} \tag{13}$$

where $u(x,t)$ is the unknown function with $x \in [-1,1]$ and $t \in [0,1]$, so $u : D \to \mathbb{R}$, with $D = [-1,1]$ x $[0,1]$. u represents the velocity or displacement of a fluid or wave in the x-direction at a given position x and time t, while ν represents the viscosity of the fluid. The equation is subject to constraints, because it has to satisfy initial conditions, given by

$$u(x,0) = -sin(\pi x), \qquad x \in [-1,1] \tag{14}$$

and the Dirichlet boundary conditions, given by

$$u(1,t) = u(-1,t) = 0, \qquad t \in [0,1]. \tag{15}$$

Physics-informed neural networks generate approximate solutions to the partial differential equation by training a neural network to minimize the loss function. Let $h(x,t;\underline{\theta})$ be the neural network: it has to be constructed with two input units, corresponding to the independent variables (x,t), multiple hidden layers and an output unit $\hat{u}(x,t)$, that is the dependent variable and corresponds to the approximation of the true solution $u(x,t)$. The vector $\underline{\theta}$ is given by the parameters of the neural network, consisting of weights and bias. It is assumed that the output of the neural network for a given input pair (x_l, t_l) represents the approximate solution of the PDE at that specific point (x_l, t_l).

The training process involves providing a sufficiently large number of input-output pairs that represent the training set, $\mathcal{T} = \{x_i, t_i, u_i\}_{i=1}^{N_\mathcal{T}}$. If the training set is sufficiently rich, from the property demonstrated in 3, the solution of the PDE can be approximated to any degree of accuracy. As it is said, to solve this complex problem, a lot of training data are required, not always available. To incorporate the known physic into the neural network architecture let's consider

$$g(\hat{u}) = \hat{u}_t(x,t) + \hat{u}(x,t)\hat{u}_x(x,t) - \nu\hat{u}_{xx}(x,t) = 0 \tag{16}$$

which follows from the equation 13. The derivatives of the approximate solution with respect to the inputs x and t, $\hat{u}_x(x_j,t_j)$, $\hat{u}_t(x_j,t_j)$, $\hat{u}_{xx}(x_j,t_j)$, are calculated via automatic differentiation. Another set of points is considered to evaluate this expression: generating a uniformly distributed set of collocation points $\mathcal{C} = \{x_j, t_j\}_{j=1}^{N_\mathcal{C}}$, compute the approximate neural network solution in these points, i.e.

$$\hat{u}_j = \hat{u}(x_j, t_j) = h(x_j, t_j; \underline{\theta}), \quad j = 1, ..., N_\mathcal{C}. \tag{17}$$

So, not considering the knowledge of the underlying dynamics, using only the training data, the loss function is given by

$$\mathcal{L}(\mathcal{C}, \underline{\theta}) = \frac{1}{N_\mathcal{T}} \sum_{i=1}^{N_\mathcal{T}} \{[\hat{u}(x_i, 0) + sin(\pi x_i)]^2 + [\hat{u}(-1, t_i)]^2 + [\hat{u}(1, t_i)]^2\}. \tag{18}$$

The mean square error loss function is chosen, that is a type of regression loss function. But then, in order to know how well the neural network h satisfies the underlying dynamics, a term encoding the physics described by the differential equation has to be added in the loss function, giving

$$\mathcal{L}(\mathcal{C}, \mathcal{T}, \underline{\theta}) = \frac{1}{N_\mathcal{C}} \sum_{j=1}^{N_\mathcal{C}} [g(\hat{u}_j)]^2 + \frac{1}{N_\mathcal{T}} \sum_{i=1}^{N_\mathcal{T}} \{[\hat{u}(x_i, 0) + sin(\pi x_i)]^2 + [\hat{u}(-1, t_i)]^2 + [\hat{u}(1, t_i)]^2\}. \tag{19}$$

We assume that the collocation points and the training points are generated by random sampling from a uniform distribution. In particular the collocation points are sampled from D, while the training data are generated by the initial and boundary conditions. The use of the mean square error as the loss function can balance the contribution from the PDE and the initial/boundary conditions.
The equation 19 shows the surprisingly simple but extremely clever idea behind PINNs. The first part of 19 represents the optimization component necessary to ensure that the neural network satisfies the given partial differential equation defined in equation 13. This part of the expression captures the discrepancy between the behavior of the neural network and the desired behavior specified by the PDE. The second part needs to be optimized to satisfy the initial/boundary conditions of the equation, given by 14 and 15.

The task of training a neural network is determining the optimal parameters $\underline{\theta}'$ and this is done by using a gradient-based optimization method, such as stochastic gradient descent or Adam optimizer.

In the figure below the method described in the section is summarized.

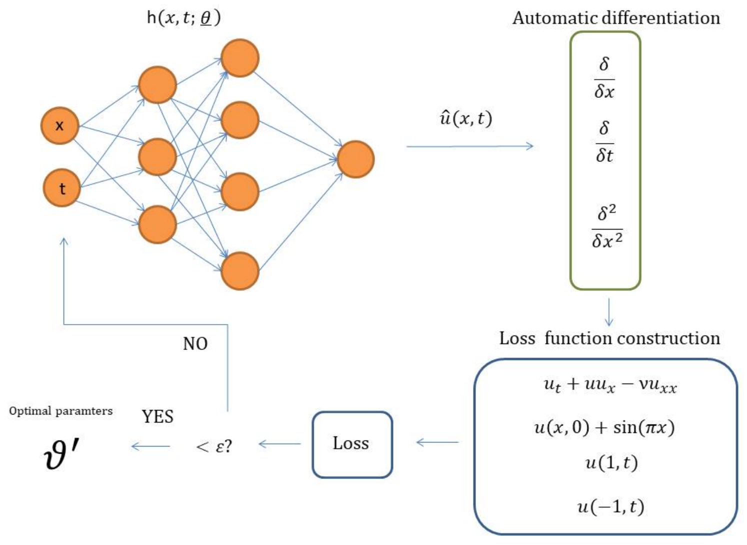

Figure 1: *Solving Burger's equation using PINNs*

5 Some critical considerations

In this last section, some critical considerations about physics-informed neural networks are discussed. Firstly, looking at the advantages of them, it can be said that the introduction of physics-informed neural networks is a great idea since using neural networks requires big data, not available in case of complex scientific problems. So incorporating the laws of physics, it is possible to utilize deep learning even with limited data. In addition, physics informed learning is capable of extrapolation, not only interpolation. PINNs have the ability to generalize well beyond the training data, meaning they can accurately predict values at locations where no training data is available. This allows for extrapolation and the ability to make predictions in unexplored regions of the input space. Then it has to be underlined the mesh-free approach: PINNs, unlike traditional numerical methods for solving differential equations, do not require explicit mesh generation or discretization of the domain, making them well-suited for problems with complex or irregular geometries. They can handle problems in arbitrary domains without the need for grid generation.

On the other hand, one disadvantage that many studies highlight[2][12] , although not specifically addressed in this paper, is the computational effort required by physics-informed neural networks compared to alternative numerical methods. However, it's important to consider that the alternative methods have benefited from years of research and development, while PINNs are a relatively recent discovery. It is possible that in the coming years, improvements in the efficiency of PINNs and optimization algorithms can help mitigate this computational challenge. With further advancements and refinements, PINNs may become more competitive with traditional numerical methods in terms of computational efficiency.

References

[1] Introduction to differential equations. ://web.uvic.ca/ tbazett/diffyqs/frontmatter-1.html.

[2] Riccardo Anelli. Physiscs-informed neural networks for shallow water equations, 2022.

[3] Harry Bateman. Some recent researches on the motion of fluids. *Monthly Weather Review*, 43(4):163–170, 1915.

[4] Johannes Martinus Burgers. A mathematical model illustrating the theory of turbulence. *Advances in applied mechanics*, 1:171–199, 1948.

[5] Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

[6] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[7] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[9] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[10] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[11] Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990.

[12] Jonas Mack. Physics informed machine learning of nonlinear partial differential equations, 2021.

[13] Andrew J Meade Jr and Alvaro A Fernandez. Solution of nonlinear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 20(9):19–44, 1994.

[14] Shagun Panghal and Manoj Kumar. Approximate analytic solution of burger huxley equation using feed-forward artificial neural network. *Neural Processing Letters*, 53:2147–2163, 2021.

[15] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 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, 2019.

[16] Walter Rudin. *Real and complex analysis.* McGraw-HiII Book Company, 1987.