Identifying Optimal Architectures of Physics-Informed Neural Networks by Evolutionary Strategy

Ana Kaplarević-Mališić[a,*], Branka Andrijević[a], Filip Bojović[a], Srđan Nikolić[a], Lazar Krstić[a], Boban Stojanović[a], Miloš Ivanović[a]

[a] *University of Kragujevac, Faculty of Science, Radoja Domanovića 12, Kragujevac, 34000, Serbia,*

Abstract

Physics-Informed Neural Networks (PINNs) are artificial neural networks that encode Partial Differential Equations (PDEs) as an integral component of the ML model. PINNs are successfully used nowadays to solve PDEs, fractional equations, and integral-differential equations, including direct and inverse problems. Just as in the case of other kinds of artificial neural networks, the architecture, including the number and sizes of layers, activation functions, and other hyperparameters can significantly influence the network performance. Despite the serious work in this field, there are still no clear directions on how to choose an optimal network architecture in a consistent manner. In practice, expertise is required, with a significant number of manual trial and error cycles. In this paper, we propose PINN/GA (PINN/Genetic Algorithm), a fully automatic design of a PINN by an evolutionary strategy with specially tailored operators of selection, crossover, and mutation, adapted for deep neural network architecture and hyperparameter search. The PINN/GA strategy starts from the population of simple PINNs, adding new layers only if it brings clear accuracy benefits, keeping PINNs in the population as simple as possible. Since the examination of dozens of neural networks through the evolutionary process implies enormous computational costs, it employs a scalable computational design based on containers and Kubernetes batching orchestration. To demonstrate the potential of the proposed approach, we chose two non-trivial direct problems. The first is 1D Stefan transient model with time-dependent Dirichlet boundary conditions, describing the melting process, and the second is the Helmholtz wave

*ana@kg.ac.rs, Radoja Domanovića 12, Kragujevac, 34000, Serbia

equation over a 2D square domain. The authors found that PINNs accuracy gradually improves throughout the evolutionary process, exhibiting better performance and stability than parallel random search and Hyperopt Tree of Parzen Estimators, while keeping the network design reasonably simple.

Keywords: PINNs, automatic design, evolutionary strategy, GA

1. Introduction

In past decades, different deep learning methods are preferred for solving various types of problems, such as image recognition, speech recognition, natural language processing, search engines, recommender systems, bioinformatics, etc. However, traditional supervised deep learning methods are not suitable for solving all types of problems, regardless of the sufficient amount of available data describing the system's behavior. Concretely, problems described by linear and non-linear equations have not been the focus of deep learning. Various numerical methods have been used for solving partial differential equations. Classical methods such as finite difference, finite volume, and finite element are considered state-of-the-art methods because of their efficiency and ability to be applied in a wide range of areas. However, solving inverse problems with classical numerical methods demands calculations that are extremely time-consuming, because dealing with unknown model parameters involves an iterative search procedure. Therefore, accuracy is often traded for efficiency. Towards eliminating these shortcomings of numerical methods, a new deep-learning method for solving partial differential equations has been developed. That method, dubbed Physics-Informed Neural Networks (PINNs) is used for solving supervised learning tasks while respecting any given law of physics described by general nonlinear partial differential equations [1]. The major innovation of PINNs compared to the classical supervised artificial neural networks (ANNs) is the introduction of a residual a network that encodes the governing physics equations, takes the output of a deep-learning network, called the approximator, and calculates a residual value [2]. The residual of the differential equation is minimized by training the approximator neural network, where differential operators are applied using automatic differentiation. To attain satisfactory generalization results in PINNs, it is crucial to choose appropriate hyperparameters and network architecture. Nevertheless, a large number of hyperparameters and their wide ranges makes it difficult to identify the combinations that provide

a good generalization performance.

Creating an optimal neural network model is an extremely difficult task. It is shown that a model's architecture and hyperparameters impact the model's performance in many ways. However, it is a particular challenge to determine hyperparameters of the model that will result in an optimal model, thus researchers are often forced to manually construct a set of hyperparameters through a repetitive and tedious process of trials and errors. Another widely used approach is a combination of grid search and manual search that dominate as the state of the art despite decades of research into global optimization [3]. In [4] it was shown that these methods can be very inefficient in large search spaces due to their exhaustive nature. Recently, Neural Architecture Search (NAS) has reached popularity in the field of automated machine learning [5]. The aim of NAS is to search for the optimal network structure in a large space of architectures, in an automated manner with minimal human involvement. The earliest NAS strategies were based on reinforcement learning methods [6, 7, 8] and were successfully used for solving image classification problems. In [9] authors achieved similar results by employing an evolutionary strategy. Both methods proved the feasibility of neural network design automation but assumed extreme computational costs. The majority of subsequent research focused on decreasing computational requirements by tailoring a search space and choosing a search strategy to reach high–performing architectures quickly. The computational cost of NAS can be significantly reduced using differentiable search methods. However, these methods often do not reach the best architectures because the best ones in the search process are not necessarily optimal for evaluation. To bridge the gap, the authors in [10, 11] present a progressive version of differentiable architecture search, where the size of the architecture gradually increases. Evolutionary algorithms have been widely used for finding an optimal topology, i.e., the number of layers and the number of neurons in each layer of an ANN [12]. These algorithms are also used as an alternative to the backpropagation algorithm for tuning the set of weights [13, 14, 15]. These hyperparameter optimizations certainly increase the performance of an ANN. However, there are other essential hyperparameters that may also affect ANN's performance. The authors in [16] propose the algorithm for the automatic design of DNNs based on evolutionary strategies not only to find the best topology but also to optimize layer-specific hyperparameters.

All these attempts refer to the classical supervised ANNs. Most NAS methods are still computationally intensive and targeted toward convolu-

tional neural networks. Typically, the PINN is a fully connected network, i.e. simple multilayer perceptron (MLP), and there are quite a few attempts at the optimization of such networks. Most of the reported PINN optimization methods refer to hyperparameter tuning for a specific type of problem. In [17] hyperparameter tuning of the PINN model for the Helmholtz operator was conducted via Gaussian processes based Bayesian optimization. The same method was used for optimizing the stochastic PINN model of the Advection-Difusion-Reaction problem in [18]. In [19] authors applied a genetic algorithm to identify the network types and optimization functions suitable for several dynamic systems. In [20] authors presented Auto-PINN, architecture, and hyperparameter optimizator for PINNs. The presented methodology implies an exhaustive search in a reduced space. The authors conducted a set of pre-experiments on PDE benchmarks for probing the structure-performance relationship, which is then used for decoupling the search of different hyper-parameters and reducing search space for each hyperparameter. The very recent research [21] explores the possibilities to optimize the architecture of a PINN, but leaving other hyper-parameters aside.

In most recent works that tackle PINN optimization, the conclusions regarding the structure-performance relationship are based on several characteristic use cases, but the question remains whether the conclusions can be generalized. Also, the network architecture is specified so that all layers of the network have the same number of neurons, making fine-tuning the complexity of the network architecture impossible. However, to the best of our knowledge, besides Auto-PINN, which is itself based on search space reduction, there is no general framework for the efficient automatic construction of PINNs.

To address this, we propose PINN/GA framework, an efficient fully automatic, and massively parallel Auto-ML solution based on a genetic algorithm (GA). Starting with an initial population of candidate solutions, called individuals, GA, as an optimization algorithm inspired by the process of natural selection, evolves the population by selecting the fittest individuals, applying crossover to create offspring with combinations of their parent's genes, and mutating some of the genes to introduce new variations in the population. When the termination criterion is reached iterative process of population evolution stops, and the fittest individual is selected. In PINN/GA, the individuals are PINNs encoded as chromosomes containing a learning algorithm, architecture, activation in hidden layers, and output activation. The operators of selection, crossover, and mutation, are also customized for neural

architecture and hyperparameter search. The PINN/GA offers two major innovations. Firstly, we keep the PINN search space as computationally simple as possible by evolutionary search from simple to more complex architectures, as proposed in [16] for general deep architectures. Secondly, we propose a scalable distributed computational design for PINN/GA optimization. With these two-fold innovations, it is possible to speed up PINN hyperparameter search by almost two orders of magnitude, making it feasible for practical use.

We demonstrate the potential of the proposed PINN/GA search method by comparing its performance with the massively parallel random search and a well-known Hyperopt's Tree of Parzen Estimators [22] on two non-trivial direct problems. The first is a direct form of the 1D transient Stefan problem. The second demo problem is a Helmholtz wave equation over a 2D square domain.

This paper is organized as follows. We first introduce the background information about PINNs, followed by a detailed description of our PINN/GA algorithm with its special operators. In Section 4, we briefly explain the practical side of the distributed computing aspects. The specifications of the benchmark cases are given in Section 5. Section 6 is all about the accuracy and performance results, followed by Conclusion with a few directions for future work.

2. Physics-Informed Neural Networks (PINNs)

The Physics-Informed Neural Network is a machine-learning technique that can be used to approximate the solution of partial differential equations. Partial differential equations (PDEs) with corresponding initial and boundary conditions can be expressed in a general form as:

$$u_t + \mathcal{N}[u] = 0, \quad X \in \Omega,\ t \in [0, T], \tag{1}$$

$$u(X, 0) = h(X), \quad X \in \Omega, \tag{2}$$

$$u(X, t) = g(X, t), \quad X \in \Omega_g,\ t \in [0, T]. \tag{3}$$

Here $\mathcal{N}$ is a differential operator, $X \in \Omega \subseteq R^d$ and $t \in R$ represent spatial and temporal coordinates respectively, $\Omega \subseteq R$ is a computational domain, $\Omega_g \subseteq \Omega$ is a computational domain of the boundary conditions, $u(X, t)$ is the solution of the PDEs with initial condition $h(X)$ and boundary condition

$g(X,t)$. A formulation like this can also be applied to higher-order PDEs, since higher-order PDEs can be written in the form of first-order PDEs.

In the original formulation [23], PINN consists of two subnets: an approximator network and a residual network. The *approximator network* receives input (X,t), undergoes the training process, and provides an approximate solution $\widehat{u}(X,t)$ of the PDEs as an output. The approximator network trains on a grid of points, called collocation points, sampled from the simulation domain. Weights and biases of the approximator network are trainable parameters that are trained by minimizing a composite loss function with the following form:

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_0 + \mathcal{L}_b, \tag{4}$$

where

$$\mathcal{L}_r = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|u\left(X^i,t^i\right) + \mathcal{N}\left[u\left(X^i,t^i\right)\right]\right|^2, \tag{5}$$

$$\mathcal{L}_0 = \frac{1}{N_0}\sum_{i=1}^{N_0}\left|u\left(X^i,t^i\right) - h^i\right|^2, \tag{6}$$

$$\mathcal{L}_b = \frac{1}{N_b}\sum_{i=1}^{N_b}\left|u\left(X^i,t^i\right) - g^i\right|^2. \tag{7}$$

Here, $\mathcal{L}_r$, $\mathcal{L}_0$, and $\mathcal{L}_b$ represent residuals of governing equations, initial and boundary conditions, respectively. Additionally, N_r, N_0, and N_b are the numbers of collocation points of the computational domain, initial and boundary conditions, respectively. These residuals are calculated by a non-trainable part of the PINN model called the *residual network*. In order to compute the residual $\mathcal{L}_r$, PINN requires derivatives of the outputs with respect to the inputs. Such computation is achieved by automatic differentiation, which relies on the fact that combining derivatives of the constituent operations by the chain rule gives the derivative of the overall composition. This technique is a key enabler for the development of PINNs and is the key element that differentiates PINNs from similar efforts in the early 90s [24, 25], which relied on the manual derivation of back-propagation rules. Nowadays, automatic differentiation capabilities are well-implemented in most deep learning frameworks such as TensorFlow [26] and PyTorch [27], and they allow us to avoid tedious derivations or numerical discretization while computing derivatives of all orders in space-time.

A schematic of the PINN is demonstrated in Figure 1 in which a simple partial differential equation $\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} = 0$ is used as an example. As shown in Figure 1, the approximator network is used to approximate the solution $u(X, t)$ which then goes to the residual network to calculate the residual loss $\mathcal{L}_r$, boundary condition loss $\mathcal{L}_b$, and initial condition loss $\mathcal{L}_0$. The weights and biases of the approximator network are trained using a custom loss function consisting of residuals $\mathcal{L}_r$, $\mathcal{L}_0$, and $\mathcal{L}_b$ through gradient-descent technique based on the backpropagation.

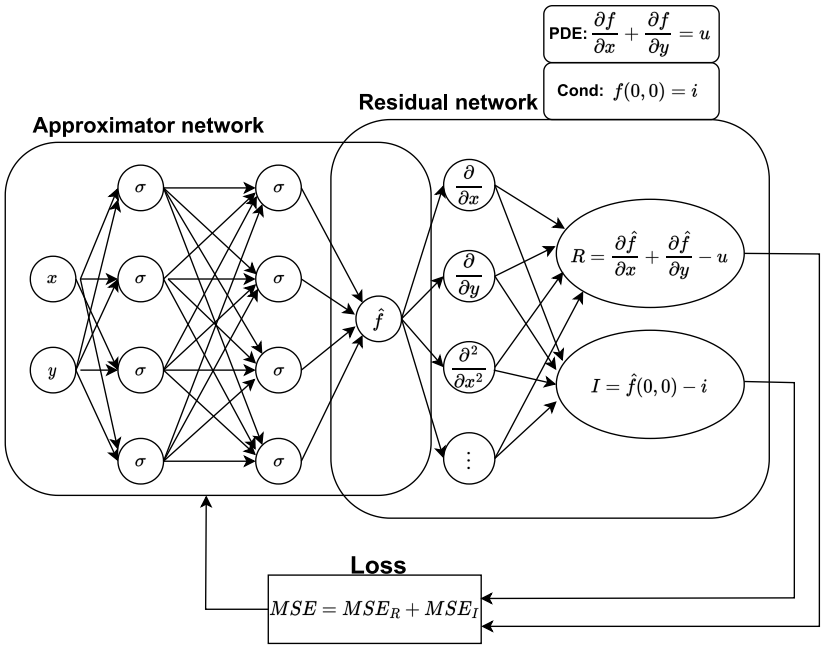


Figure 1: The architecture of the PINN and the standard training loop of PINN constructed for solving a simple partial differential equation, where PDE and $Cons$ denote governing equations, while R and I represent their residuals. The approximator network is subjected to a training process and provides an approximate solution. The residual network is a non-trainable part of PINN capable to compute derivatives of approximator network outputs with respect to the inputs, resulting in the composite loss function, denoted by MSE.

3. PINN/GA optimization method

There are no clear guidelines on how to design an ANN model, making the design process substantially complex. Trial and error approach for validation

of models with different architectures and hyperparameters requires a lot of time. This problem also occurs in PINN design, since the only difference compared to classical neural networks is that PINNs are trained to satisfy any given law of physics instead of satisfying a predefined output. We propose an automatic approach that employs a specially tailored evolutionary strategy to search for optimal architecture and hyperparameters of a PINN, while the weights of individual PINNs are learned by classical gradient-based technique.

3.1. Genetic algorithm for a PINN optimization

Genetic algorithms (GAs) are evolutionary algorithms that have proven to be powerful and robust mechanism when it comes to solving complex, realistic optimization problems. GA rests on the theory of natural selection where the best individuals are chosen to survive and produce offspring for the next generation. In each generation, a new set of individuals is created using pieces of the individuals from the previous generation. In GA terminology, each individual is represented by a chromosome which is a potential solution to a problem. The chromosomes are made up of genes that contain a part of genetic material, and each gene controls one or more chromosome characteristics. GA processes a population of chromosomes through generations using a kind of natural selection through the genetics-inspired operators of selection, crossover, and mutation [28].

The process starts with a randomly generated population. All individuals of the initial population are evaluated to obtain their fitness. The population is then subjected to the iterative process of selection, crossover, mutation, and evaluation for the next iteration (generation). The selection operator chooses those individuals from the population that will be allowed to reproduce, and, on average, the fitter individuals, called parents, produce more offspring than the less fit ones. Crossover exchanges sub-parts of two-parent individuals [28], whereby there is an exchange of genetic material. Constant application of selection and crossover operations leads the population to an increasing number of individuals with good genes, approaching the global optimum. The mutation operator introduces stochastic changes in the characteristics of chromosomes, for the purpose of reintroducing genetic diversity back into the population and assisting the search to escape from local optima [29]. This iterative process stops when the termination criterion is reached.

The proposed PINN/GA optimization methodology is based on the iterative strategy of GA, shown in Figure 2. In our approach to identifying

the optimal PINN architecture and hyperparameters, each individual represents an individual PINN model. All individuals of the current population are evaluated to determine their fitness, taken as a measure of satisfying the governing differential equation, initial, and boundary conditions. The individual with a lower mean squared error (MSE) is considered fitter. According to the chosen selection criterion and the elitism concept, only the fitter individuals enter the reproduction phase. Using the operators of crossover and mutation, new generations of PINNs with various architectures and hyperparameters are created. Once the stop criterion is achieved, the output is a population of the fittest individuals. The stop condition can be a predefined number of generations, or a hyper-volume indicator [30]. Upon the optimization process completion, an individual with the lowest MSE is chosen as the optimal one.

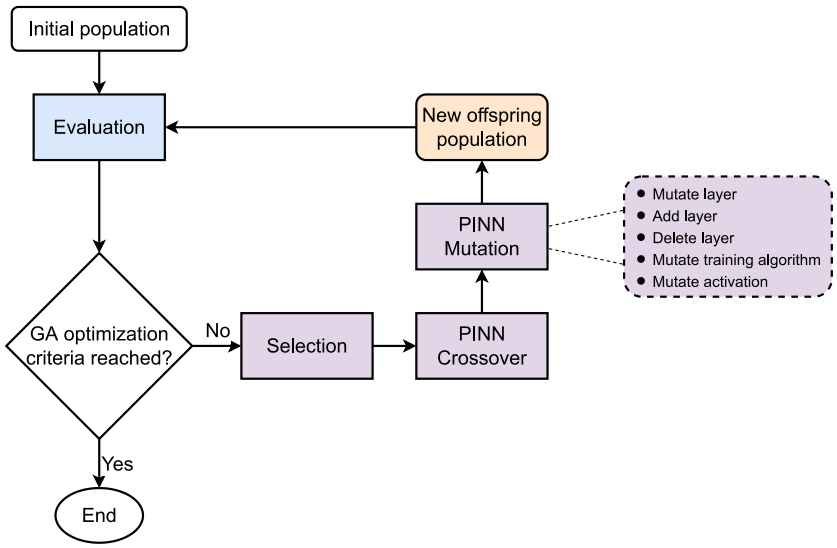


Figure 2: PINN/GA evolutionary loop. Implements an iterative GA strategy, where each individual represents a PINN. Special crossover and mutation operators aim at better PINN performance while keeping PINNs' architecture as simple as possible.

3.2. PINN chromosome structure

Let $PINN^{(\psi)}$ denote an individual PINN from the set of all possible PINNs ω. For finding an optimal PINN architecture, we propose a specific GA approach with the genetic structure of chromosomes shown in Figure 3. Each chromosome encodes a learning algorithm, architecture of the PINN

model, activation function in the hidden layers, and output activation. A similar chromosome structure is originally introduced in [16], where an optimal architecture of a classical deep neural network was the subject of the search.

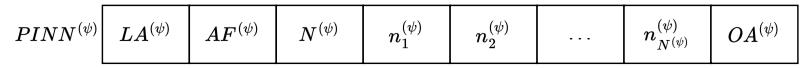


Figure 3: The structure of the PINN/GA chromosome. The chromosome encodes: the learning algorithm, $LA^{(\psi)}$; the activation function, $AF^{(\psi)}$, common for all hidden layers; the number of hidden layers, $N^{(\psi)}$; the number of neurons in each hidden layer, $n_1^{(\psi)}, n_2^{(\psi)}, \ldots, n_{N^{(\psi)}}^{(\psi)}$; the output activation, $OA^{(\psi)}$.

The very first gene $LA^{(\psi)}$ represents the learning algorithm. The next gene, $AF^{(\psi)}$, represents the activation function acting in all hidden layers. The gene $N^{(\psi)}$ represents the number of hidden layers. Next $N^{(\psi)}$ genes characterize specific hyperparameters for each layer. These genes can contain one or more sub-genes that act as layer-specific hyperparameters. In this chromosome interpretation, only the number of neurons in hidden layers are stored in genes $n_1^{(\psi)}, n_2^{(\psi)}, \ldots, n_{N^{(\psi)}}^{(\psi)}$. The last gene, $OA^{(\psi)}$, denotes the output activation. For each of the previously mentioned elements of the PINN chromosome, we must specify ranges representing the search space. PINNs can contain up to $N_{\max}$ hidden layers and each hidden layer contains up to $n_{\max}$ neurons. All neurons from all hidden layers use the same activation function from the set of possible activation functions AF. The same goes for the neurons in the output layer, but the specific kind is taken from the set of possible output activations OA. Additionally, each model picks one of the offered learning algorithms LA [29]. The possible values of each gene of the chromosome are defined as:

$$LA^{(\psi)} \in \{Adam,\ RMSprop,\ Adagrad,\ Nadam\}, \tag{8}$$

$$AF^{(\psi)} \in \{tanh,\ sigmoid,\ selu,\ softmax,\ relu,\ elu,\ sin\}, \tag{9}$$

$$1 \leq N^{(\psi)} \leq N_{\max}, \tag{10}$$

$$1 \leq n_i^{(\psi)} \leq n_{\max}, \quad i = 1,\ \ldots,\ N^{(\psi)}, \tag{11}$$

$$OA^{(\psi)} \in \{linear,\ relu\}. \tag{12}$$

All variables except the number of hidden layers are encoded as an integer values that represent the index of an element from the corresponding set of

possible values. The error of the $PINN^{(\psi)}$ model ψ is mean squared error calculated as:

$$MSE^{(\psi)} = max\left\{MSE^{(\psi)}_{train},\ MSE^{(\psi)}_{val}\right\}, \tag{13}$$

where $MSE^{(\psi)}_{train}$ and $MSE^{(\psi)}_{val}$ represent PINN's mean squared error on training and validation set, respectively. This way, the model's ability to generalize is increased. The best choice of architecture and hyperparameters can be written briefly as **min** $\left\{Minimize\ MSE^{(\psi)} \mid \forall\ PINN^{(\psi)} \in \omega\right\}$.

3.3. Initial population

The initial population consists of individuals whose genes take random values from the corresponding set of possible values (8)-(12), except for the gene that represents the number of hidden layers $N^{(\psi)}$. Following the approach proposed in [16], the number of hidden layers for all individuals in the initial population is set to $N^{(\psi)} = 1$. The number of hidden layers can only be modified as a result of the mutation operator, which will be explained later. If an increased number of hidden layers brings accuracy improvement, the PINN/GA algorithm will guide the optimization process toward increasing the number of layers in the entire population. This controlled increase in complexity results in an algorithm that converges faster since simpler PINNs require less training time.

3.4. PINN/GA operators

We cover the operators of selection, crossover, and mutation, each specially tailored for neural architecture and hyperparameter search.

3.4.1. Selection

The better the individual, the greater the chance of being chosen as a parent. This creates selection pressure that leads the population to better solutions. According to the optimization criterion, the individuals are ranked based on fitness value (13). To ensure the survival of the best individuals, the elitism strategy has also been used [31]. This means that two fittest individuals from the current generation pass directly to the next generation. To determine the remaining individuals for the next generation, we apply binary tournament selection, which performs a tournament between a randomly chosen pair of individuals from the current generation. Fitter individuals from each pair enter the reproduction phase, where crossover and mutation operators are applied.

3.4.2. Crossover

The crossover operator combines the genes of two parent individuals to produce two offspring individuals. It is implemented as a one-point crossover. Suppose that two parents P_1 and P_2 are given as

$$P_1 = \left(LA^1,\ AF^1,\ N^1,\ n_1^1,\ n_2^1, \ldots,\ n_{N^1}^1,\ OA^1\right), \tag{14}$$
$$P_2 = \left(LA^2,\ AF^2,\ N^2,\ n_1^2,\ n_2^2, \ldots,\ n_{N^2}^2,\ OA^2\right). \tag{15}$$

Here, N^1 and N^2 are numbers of hidden layers of parents P_1 and P_2, respectively. The cross-point divides individuals into two parts: a head and a tail. As shown in [16], the cross-point can only be located between two genes representing the number of neurons in a hidden layer. The crossover application over parents P_1 and P_2 produces offspring

$$O_1 = \left(LA^1,\ AF^1,\ N^2,\ n_1^1, \ldots,\ n_{cp}^1,\ n_{cp+1}^2, \ldots,\ n_{N^2}^2,\ OA^2\right), \tag{16}$$
$$O_2 = \left(LA^2,\ AF^2,\ N^1,\ n_1^2, \ldots,\ n_{cp}^2,\ n_{cp+1}^1, \ldots,\ n_{N^1}^1,\ OA^1\right), \tag{17}$$

where cp is the cross-point with randomly selected value from the set $\{1, \ldots, min(\ N^1,\ N^2) - 1 \mid N^1,\ N^2 > 1\}$. Those individuals exchange their tails, together with genes containing the number of hidden layers. The crossover operation is presented in Figure 4.

3.4.3. Mutation

The mutation is the unary operator representing a small change in an individual's genetic material. Mutation provides an escape from the local optimum and is a special highlight of the PINN/GA approach. The the mutation operator is applied under a certain probability and each of the following types of mutations have an equal probability of being elected:

- *Mutate layer* – the hidden layer mutation acts on a randomly selected hidden layer by replacing the number of neurons with a randomly chosen number of neurons from a specified range.
- *Add layer* – the number of hidden layers increases when a new layer with a random number of neurons appends to an individual PINN. This mutation operator can be performed only if the limit of a maximum number of layers is not violated.

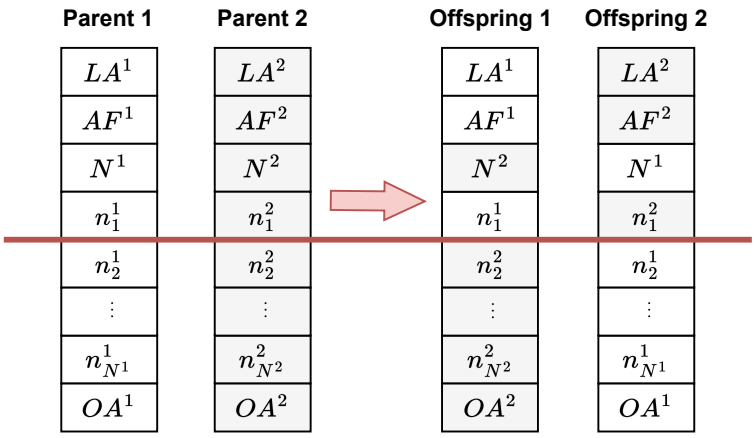


Figure 4: PINN/GA crossover operator. The individuals exchange their tails, along with genes encoding the total number of the hidden layers. Chromosome heads are positioned above the red line, tails being below.

- *Delete layer* – removal of a randomly chosen layer from the individual. This mutation operator can be performed only if there are at least two hidden layers in a selected PINN.
- *Mutate training algorithm* – a training algorithm is chosen from a set of possible training algorithms, reflecting the change of the *optimizer* gene OA.
- *Mutate activation* – choose from a set of possible activation functions, reflecting the change of the *activation* gene.

3.5. Evaluation of individuals

Each individual from the population impersonates the architecture and hyperparameters of a specific PINN. The evaluation of each individual implies building a PINN with a combination of hyperparameters and architecture represented by the chromosome. To obtain a valid result, all PINNs exhibit training on an identical set of collocation points, uniformly sampled from the computational domain. The result of the evaluation is the individual's fitness, represented by MSE (13).

4. Distributed implementation of PINN/GA search

Since the examination of dozens of PINNs through the evolutionary process implies enormous computational costs, to validate the proposed concept, we developed a distributed software framework based on contemporary technologies.

PINN/GA makes use of a slightly modified WoBinGO framework [32, 33, 34], shown in Figure 5. Its major feature is the distributed evaluation of the population of PINNs, capable of training multiple PINNs at the same time, ideally the entire generation. The PINN/GA framework's scalable computational design is based on containers and Kubernetes batching orchestration, significantly reducing the time for training multiple PINNs simultaneously. The optimization process is initiated and can be monitored through *JARE Manager*. It interacts with *JARE* service, which oversees the guidance of the whole optimization process. The framework operates according to a simple manager-worker principle. Whenever a generation of PINNs needs to be evaluated, the JARE service creates an array of asynchronous Kubernetes batching requests. The main PINN/GA loop then awaits all requests to complete, gathering fitness values for all PINNs. *Binder master* acts as a helper service capable of storing various PINN models and making them available for the worker pods. It makes use of *rsync* protocol for the sake of efficiency. In each PINN/GA generation, upon evaluating all PINNs, *JARE* applies selection, crossover, and mutation operators described in Section 3.4 to obtain the next generation. Subsequently, the new generation is sent for the distributed evaluation, and the whole process repeats until the GA stopping criterion is reached (Figure 2).

It is important to mention that we use this same framework to implement the parallel random search with which we compare the proposed PINN/GA approach since the JARE component implements a simple random search, as well.

5. Case studies

The proposed PINN/GA approach was validated in two distinct case studies. The first is the 1D Stefan problem of the ice melting process, and the second is a 2D standing acoustic wave modeled by the Helmholtz equation.

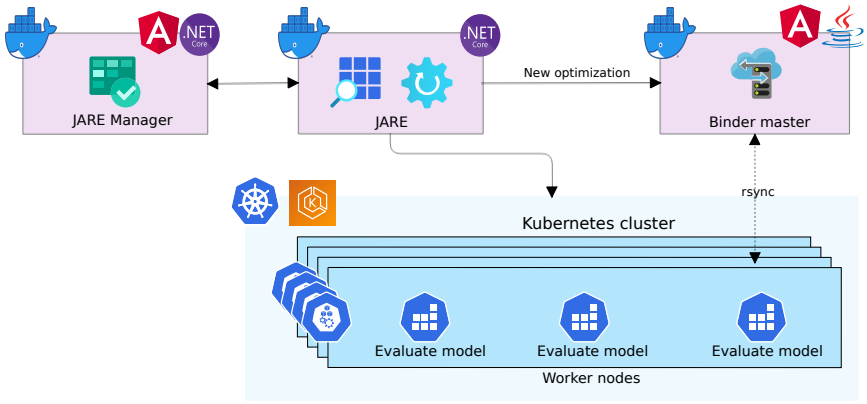


Figure 5: PINN/GA distributed software framework. *JARE Manager* initiates and monitors the optimization process interacting with *JARE service*. Whenever an evaluation of PINNs is needed, *JARE service* creates an array of asynchronous Kubernetes batching requests. *Binder master* stores PINN models and delivers them to the worker pods for evaluation. The main PINN/GA loop awaits to gather the fitness values of all PINNs. If stopping criteria is not yet reached, *JARE service* creates a new generation of PINNs and the process repeats.

5.1. Stefan problem in 1D

Stefan problems with the change of phase have applications in various fields of science and industry, in which the phase changes from liquid, solid, or vapor states. The material is assumed to undergo a phase change with a moving boundary whose position is unknown and must be determined as a part of the analysis. Since moving boundary problems require solving the heat equation in an unknown region that also has to be determined as a part of the solution, they are inherently non-linear. One dimensional phase change problem could be demonstrated by a semi-infinite solid, like a thin block of ice occupying $0 \leq x < \infty$, on solidification temperature. At the fixed boundary of the thin block of ice ($x = 0$), various types of flux functions could act. In this case study, we use the same boundary condition as [35, 36], so the temperature at $x = 0$ increases exponentially with time. We also expect that the entire solid phase has a melting point temperature. Therefore, the problem is to determine the temperature distribution in the liquid phase at time t_0, where $x < s(t_0)$, as well as the location of the free boundary $s(t_0)$. At a later time $t_1 > t_0$, the moving boundary $s(t)$ moves to the right and occupies the position $s(t_1) > s(t_0) = s_0$, as shown in Figure 6. The part of the thin block of ice from position $s(t_0)$ to position $s(t_1)$, has

melted over the time interval (t_0, t_1).

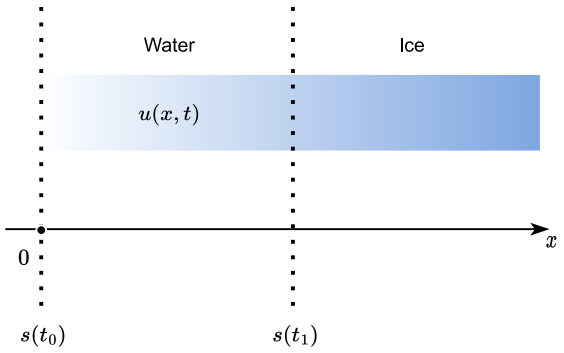


Figure 6: 1-D Stefan problem. $s(t)$ denotes the moving boundary and $u(x,t)$ the liquid phase $(x < s)$ temperature.

The temperature distribution $u(x,t)$ in the liquid phase region $0 \leq x \leq s(t)$ is given by the heat equation:

$$\frac{\partial u}{\partial t} = \alpha \cdot \frac{\partial^2 u}{\partial x^2}, \tag{18}$$

that can be written in this form:

$$\frac{\partial u}{\partial t} - \alpha \cdot \frac{\partial^2 u}{\partial x^2} = 0, \tag{19}$$

under the following boundary conditions:

$$u(x,t) = e^{\alpha t}, \quad x = 0, \quad t > 0 \tag{20}$$

$$u(x,t) = 1, \quad x = s(t), \quad t > 0. \tag{21}$$

Here, α denotes a physical parameter combining thermal conductivity, density, and specific heat. The location of the moving boundary is given by the equation known as Stefan condition:

$$\frac{1}{\alpha} \cdot \frac{ds}{dt} = -\frac{\partial u}{\partial x}, \quad x = s(t), \quad t > 0. \tag{22}$$

In the general case, the initial condition is given by

$$s(0) = 0. \tag{23}$$

The solution to this problem is known as:

$$u(x,t) = e^{\alpha t - x} \tag{24}$$

$$s(t) = \alpha t. \tag{25}$$

Solving this problem following the PINN approach assumes constructing two neural networks. The first one approximates the temperature distribution function $u(x,\ t)$ and the second approximates the function of free boundary location $s(t)$. Approximate solutions are differentiated with respect to their variables, for values defined in the set of collocation points selected from the domain $[0, T] \times \mathcal{D}$, where $\mathcal{D} \subset \mathbb{R}^d$ is a bounded domain, and T denotes the final time. Loss function consists of terms used in (19), (20) and (21) by neural network approximations of u and s at collocation points, where terms include a given partial differential equation and the initial and boundary conditions along the domain boundary.

The first approximator network approximates the temperature distribution function $u(x,t)$, and the second approximates the function of free boundary location $s(t)$. The loss function consists of terms used for assessment difference between u and s and their approximations $\widehat{u}$ and $\widehat{s}$ given by a PINN. Here, terms represent residuals of the governing partial differential equation, the initial and boundary conditions. The total loss $\mathcal{L}$ is determined by the sum of residuals as:

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_0 + \mathcal{L}_{b_1} + \mathcal{L}_{b_2} + \mathcal{L}_{b_3}, \tag{26}$$

where

$$\mathcal{L}_r = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|\frac{\partial\widehat{u}(x,t)}{\partial t} - \alpha\frac{\partial^2\widehat{u}(x,t)}{\partial x^2}\right|^2, \tag{27}$$

$$\mathcal{L}_0 = \frac{1}{N_0}\sum_{i=1}^{N_0}\left|\widehat{s}(0) - s(0)\right|^2, \tag{28}$$

$$\mathcal{L}_{b_1} = \frac{1}{N_{b_1}}\sum_{i=1}^{N_{b_1}}\left|\frac{1}{a}\frac{\partial\widehat{s}(t)}{\partial t} + \frac{\partial\widehat{u}}{\partial\widehat{s}(t)}\right|^2, \tag{29}$$

$$\mathcal{L}_{b_2} = \frac{1}{N_{b_2}}\sum_{i=1}^{N_{b_2}}\left|\widehat{u}(0,t) - u(0,t)\right|^2, \tag{30}$$

$$\mathcal{L}_{b_3} = \frac{1}{N_{b_3}}\sum_{i=1}^{N_{b_3}}\left|\widehat{u}\left(\widehat{s}(t),t\right) - u\left(s(t),t\right)\right|^2. \tag{31}$$

The first term $\mathcal{L}_r$ penalizes the governing equation (19), N_r being the batch size of collocation points randomly sampled in the training domain consisting of spatial and temporal coordinates taking values from $0 \leq x \leq 1$ and $0s \leq t \leq 0.5s$, respectively. $\widehat{u}(x,t)$ is the neural network approximation of the temperature field $u(x,t)$. The second term $\mathcal{L}_0$ determines the fulfillment of the initial condition (23). The fulfillment of the Stefan condition (22) is given by residual $\mathcal{L}_{b_1}$, $\widehat{s}(t)$ denoting PINN approximation of the moving boundary position. The last two terms $\mathcal{L}_{b_2}$ and $\mathcal{L}_{b_3}$ indicate residuals of boundary conditions (20) and (21) N_0, N_{b_1}, N_{b_2}, and N_{b_3} denote the numbers of the collocation points in which initial and boundary conditions apply.

5.2. Helmholtz equation over a 2D square domain

The Helmholtz equation is an important tool in the field of acoustics to study the behavior of acoustic waves in different types of media. It describes the behavior of the pressure wave in space, by modeling sound in the frequency domain.

We choose to model a standing wave in a 2D domain of a simple unit square, where a sound signal is assumed to be time-harmonic, as described by [37] and Lu Lu[1]. The problem of pressure change can be modeled by the

[1]https://deepxde.readthedocs.io/en/latest/demos/pinn_forward.html

following Helmholtz equation:

$$\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - k_0^2 u = f, \quad \Omega = [0,1]^2, \tag{32}$$

where k_0 designates the wave number with a value of $k_0 = 2\pi n$. We choose the multiplier $n = 2$, and the source term f as $f(x,y) = k_0^2 \sin(k_0 x)\sin(k_0 y)$, under the following Dirichlet boundary conditions:

$$u(x,y) = 0, \quad (x,y) \in \partial\Omega, \tag{33}$$

as shown in Figure 7.

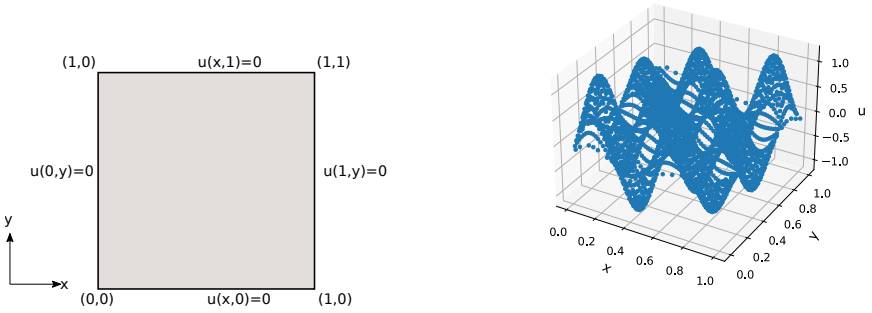


Figure 7: (a) The 2D unit square domain for solving Helmholtz equation with prescribed Dirichlet boundary conditions. (b) The exact solution to the posed problem.

The analytical solution to this problem is

$$u(x,y) = \sin(k_0 x)\sin(k_0 y) \tag{34}$$

Solving this problem following the PINN approach assumes constructing a neural network that approximates pressure distribution function $u(x,y)$. The approximate solution is differentiated with respect to its variables, for values defined in the set of collocation points selected from bounded domain $\Omega = [0,1]^2$. The loss function consists of terms used in (32), and (33) by neural network approximation of u at collocation points, where terms include given partial differential equation and boundary conditions along the domain boundary:

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_{b_1} + \mathcal{L}_{b_2} + \mathcal{L}_{b_3} + \mathcal{L}_{b_4}, \tag{35}$$

where $\mathcal{L}_r$ represents residual that originate from the PDE (32), and $\mathcal{L}_{b_i}$ residuals computed by Dirichlet boundary conditions (33) for each side of the square.

The choice of the Helmholtz equation for the PINN/GA benchmark is deliberate. This type of problem is still a challenge for many numerical methods including FEM, from the point of view of formulation, mesh density, etc. When solving by the PINN approach, special attention must be paid to the density of collocation points (at least 10-30 per wavelength), the choice of activation functions and architecture. A small change can lead to a lack of convergence.

6. Results and Discussion

In order to evaluate the quality, performance, and stability of the PINN/GA approach, we compare the real-world performance of the proposed method with two other highly parallelizable methods. The first is the Random search, while the second is the widely used Hyperopt Tree of Parzen Estimators [22]. To ensure identical conditions, all algorithms run on the same computing cluster. PINN/GA utilizes the platform described in Section 4, while Random search and Hyperopt TPE rely on parallelization provided by Apache Spark. To complete the experimental setup description, the details of PINN implementation and the parameters of the GA are given in the next two paragraphs, respectively.

We employed SciANN library [38] to implement both models (Stefan and Helmholtz). SciANN is a Python package for physics-informed deep learning using ANNs, based on the Keras and TensorFlow back-ends. The meshes of collocation points in both models are regular 100×100 grids, 70% of which is used for training, and the remaining 30% for validation. The batch size was set to 512 and the learning rate to 0.02. The training of models lasts for 3000 epochs for all PINNs considered throughout the optimization process for all competing optimization methods.

As for the PINN/GA search, we set the following parameters. Each generation consists of 40 individuals and the whole GA optimization process lasts for a maximum of 100 generations, or less if the hyper-volume indicator reaches the threshold. The mutation and crossover probabilities are set to 0.4 and 0.9, respectively. It should be noted that the mutation probability is higher than usual to give the opportunity to all important special operators described in Section 3.4.3 to act. The upper limits for the number of hidden

layers and the number of neurons per layer are 10 and 30, respectively, while activation, output activation, and learning algorithms are chosen from the sets specified in (8)-(9). We set the identical ranges for Random search and Hyperopt optimization methods.

Since the major aim of the conducted research is to demonstrate the benefits of doing optimization in the distributed computing environment in a smart way, it is also important to describe the underlying computing platform. All experiments were carried out on a Kubernetes 1.22 cluster consisting of 7 physical nodes. Each node is equipped with dual Intel Xeon E5-2683 v4 @ 2.1GHz CPU (32 physical cores), 128GB memory, and 10 Gb interconnection, totaling 224 cores and 896GB of RAM. The base OS platform is CentOS 7.9 x86_64.

6.1. 1D Stefan problem

Figure 8 demonstrates the entire flow of the optimization process for the 1D Stefan problem for Random search, Hyperopt TPE, and proposed PINN/GA. The left axis and corresponding solid lines show the fitness of the best individual at a current time defined as (13). The right axis and dotted plots show the best individual's complexity in terms of trainable parameter count (TPC). All plots are derived as an average from 10 distinct runs, which took approximately 20 hours in total.

At the very beginning, the Random search finds a very fit, but solidly complex network and performs better than both Hyperopt and PINN/GA, but without systematic improvement over time. Hyperopt TPE also exhibits a good gradient at the beginning and finishes slightly better than Random search. On the other hand, PINN/GA gradually improves by adding more layers and stays better than both competing methods almost all the time, while keeping PINNs reasonably simple. It reaches a steady state after approximately 2 hours. As explained in Section 3, due to the specific implementation of the mutation operator, we set all PINNs of the initial population to exactly one hidden layer. As PINN/GA progresses, we can observe how models become more complex over generations. Sometimes, the complexity reduces as a result of the *Delete layer* operator, acting as a certain kind of regularization mechanism. The fitness represented by $log(MSE)$, and the PINN complexity represented by Trainable parameter count (TPC), reach the plateau almost simultaneously. Another interesting finding is the indirect demonstration of the validity of the *Observation 1* found in [20]. It says

that there is a dominant PINN activation for a certain problem. Judging by the final GA population, 92% of all individuals possess $Tanh$ activation.

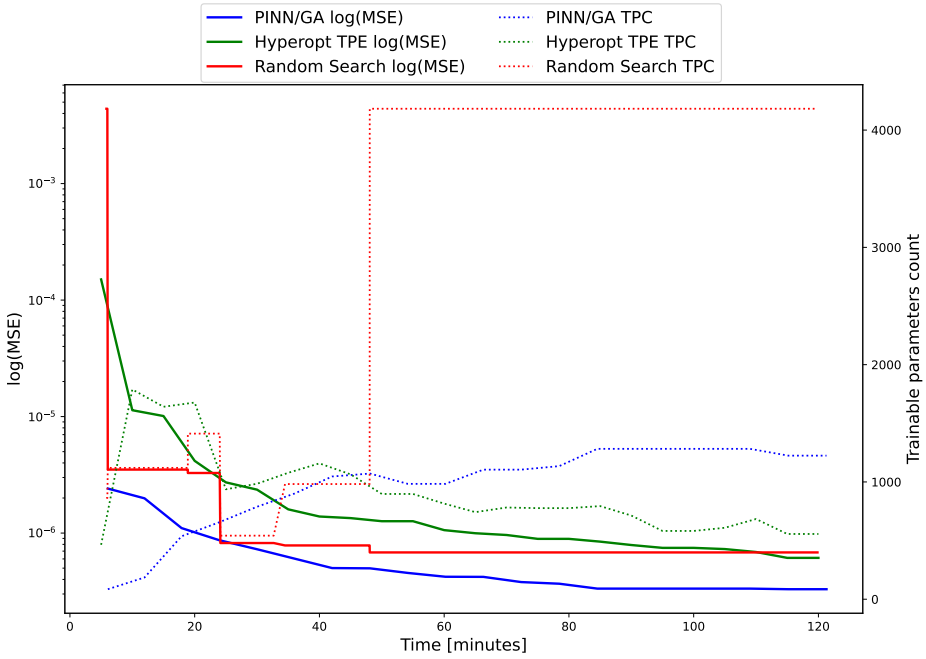


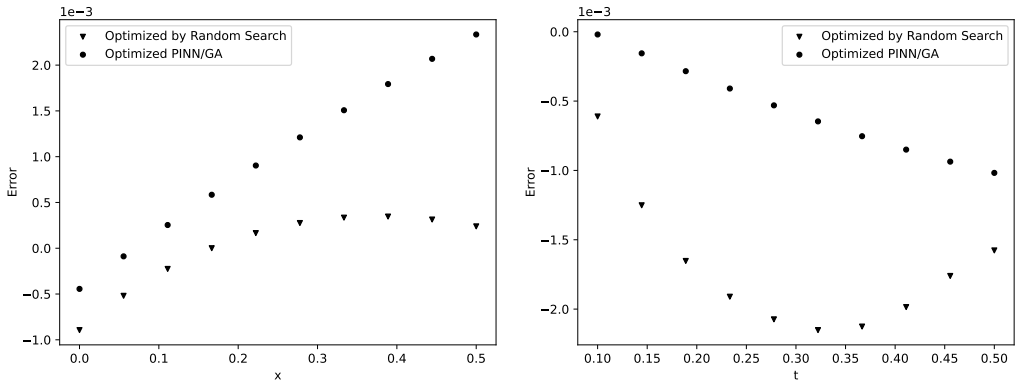
Figure 8: The PINN hyperparameter optimization process for Stefan's problem. The left axis and corresponding solid lines show the fitness of the currently best PINN. The right axis and dotted plots show the best PINN's complexity.

The optimal hyperparameters for the final PINNs approximating the one-dimensional Stefan problem of ice melting determined by all three optimization methods are enlisted in Table 1. It should be emphasized that these are the best individuals obtained out of all 10 distinct runs. Only PINN/GA found a very simple, but very fit PINN.

Table 1: The optimal PINN hyperparameters for 1D Stefan problem obtained by PINN/GA, Random search, and Hyperopt TPE, taken from 10 independent runs.

Hyperparameter	**PINN/GA**	**Random search**	**Hyperopt TPE**
Optimizer	`Nadam`	`Nadam`	`Nadam`
Activation function	`Tanh`	`Tanh`	`Sigmoid`
Number of hidden layers	2	8	7
Neurons in hidden layers	[3, 22]	[3, 12, 16, 10, 10, 18, 18, 18]	[3, 25, 21, 5, 4, 28, 13]
Output activation	`Linear`	`Linear`	`Linear`

Finally, we compare the performance of an optimal PINN obtained by PINN/GA (best) with the results of the PINN obtained by Random search (worst). Figure 9a depicts residuals of the temperature field u, while Figure 9b shows residuals of the predicted positions of the moving boundary s. While the prediction of u is slightly worse for PINN/GA optimized model, it exhibits significantly better accuracy in terms of the moving boundary s. Comparing the topological complexities of optimized PINNs, listed in Table 1, it is obvious that PINN/GA is significantly less complex than optimal PINNs obtained by Random search and Hyperopt, thanks to the specific gradual strategy and the elitism coded in GA. Less complex architectures exhibit better performance in inference tasks.



(a) Temperature field u at $t = 0.5$

(b) Position of the moving boundary s

Figure 9: Stefan problem - residual plots of optimal PINNs obtained by PINN/GA and Random search specified in Table 1.

6.2. Helmholtz equation over a 2D domain

Figure 10 shows the flow of the optimization process for the acoustic wave propagation problem described by the Helmholtz equation in Section 5.2. As for the previous use case, the plots for PINN/GA and Hyperopt are derived as an average from 10 independent runs. Both fitness of the best individuals (solid lines) and Trainable Parameter Count (dotted lines) exhibit similar trends as in the case of Stefan's problem shown in Fig. 8, but give more interesting insights. At the very beginning, the Random search again performs much better than PINN/GA and Hyperopt, thanks to a lucky hit of a high-performance, but also modestly complex network. Hypeopt advances

faster than PINN/GA thanks to the availability of the entire search space, while PINN/GA gradually adds layers and monitors its impact on performance. Due to this systematic nature of the proposed approach, PINN/GA surpasses Random search by around 40. minute, Hyperopt by 50. minute, and stays constantly better until the very end of the optimization process.

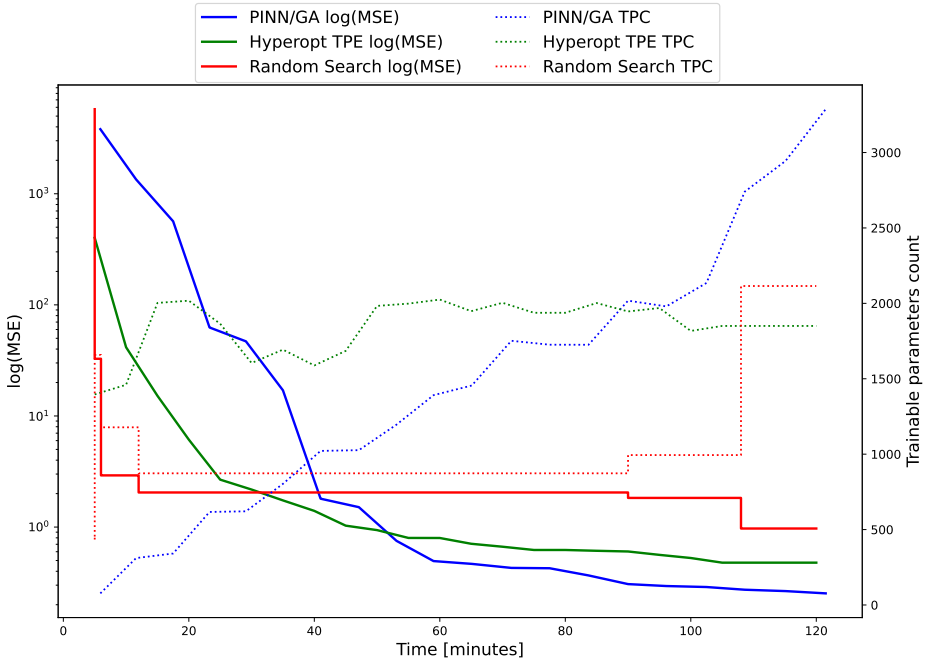


Figure 10: The PINN hyperparameter optimization process for Helmholtz problem. The left axis and corresponding solid line plots show the fitness of the best individual. The right axis and dotted plots show the best individual's complexity.

The optimal hyperparameters for the final PINNs obtained from all 10 independent runs are listed in Table 2. The learning rate, the batch size, and the number of epochs are all constant, as specified above. It is apparent that PINN/GA keeps adding complexity until the very end of the optimization process, reaching the maximum of 30 neurons per layer in all layers. Moreover, the trend of the $log(MSE)$ curve in Fig. 10, clearly indicates that a further increase in complexity would probably bring further benefit to the accuracy. If we take a closer look at the proposed architecture, all layers are full, but the number of layers (5) is less than the maximum prescribed by GA (10). Therefore, one should probably search in the direction

of fewer layers with a larger number of neurons than the current limit. This recommendation is consistent with the findings of [17], who suggests quite shallow architectures, with depth $2, 3$, and high width (> 150) for such wave problems. Even if in the end one doesn't get a production-quality PINN, the PINN/GA method indirectly recommends in which direction we should move. The *Observation 1* from [20] seems true in the Helmholtz case as well, since 100% of the individual PINNs in the final generation possess *sin* activation.

Table 2: The optimal PINN hyperparameters for the Helmholtz problem obtained by PINN/GA, Random search, and Hyperopt TPE, taken from 10 independent runs.

Hyperparameter	**PINN/GA**	**Random Search**	**Hyperopt TPE**
Optimizer	`Nadam`	`Adam`	`Adam`
Activation function	`Sin`	`Sin`	`Sin`
Number of hidden layers	5	5	`3`
Neurons in hidden layers	$[30, 30, 30, 30, 30]$	$[22, 13, 29, 25, 23]$	$[17, 17, 3]$
Output activation	`Linear`	`Linear`	`Linear`

At the end of the section, a few words about the performance of the distributed software framework. Since the ideal speed-up equals the size of the population in GA, set to 40, the achieved speed-up of approximately 27 is a satisfactory result, having in mind the load diversity due to variable training times of PINNs with various complexities.

7. Conclusion

In this paper, a novel approach is presented identifying an optimal combination of architecture and other hyperparameters of PINNs using an evolutionary strategy. The proposed PINN/GA technique uses the genetic algorithm with specially tailored operators of selection, crossover, and mutation. The optimization process keeps the PINNs population as simple as possible since new network layers are added only if they bring clear benefits. To make the optimization feasible in a normal time frame, PINN/GA hyperparameter search runs on a distributed computing framework based on Kubernetes container orchestration.

PINN/GA approach was validated in two distinct use cases. The first is the 1D Stefan problem, with time-dependent Dirichlet boundary conditions driving the ice-melting process. The second is an acoustic wave forming in a 2D domain respecting the Helmholtz equation. The experimental verification of the PINN/GA method showed promising results in terms of accuracy

and computational efficiency. Through the optimization process, the average error gradually decreased by an order of magnitude, exhibiting predictability and constant improvement throughout the optimization process. The optimized PINN/GA model provides visibly better accuracy than PINN optimized by a parallel random search and Hyperopt's TPE method carried out in the same distributed computing environment. In most cases, PINN/GA keeps the network topology reasonably simple. Even if the optimization process reaches the limits posed by the GA setup, PINN/GA provides useful information in which direction the search should be going. From the strict user's point of view, one should decide whether waiting for approximately 1-2 hours to get the optimal combination of hyper-parameters instead of 6-7 minutes for a PINN with random hyper-parameters is worth the effort.

In the future, the authors plan to extend the proposed PINN/GA method so multiple layer-specific hyperparameters can be optimized simultaneously, opening the possibility of finding even more favorable architectures. In order to bring the use of the proposed methodology closer to the researchers who are not primarily engineers, future work will also focus on developing an open-source framework for the automation of the whole PINN optimization process. Such a framework should provide an interface to the well-known PINN libraries such as SciANN [38] and DeepXDE [39], enabling researchers to optimize an already built PINN by a single function call.

Acknowledgement
This paper is funded through the EIT's HEI Initiative SMART-2M project, supported by EIT RawMaterials, funded by the European Union.

References

[1] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations, CoRR abs/1711.10561 (2017).

[2] S. Markidis, The old and the new: Can physics-informed deep-learning replace traditional linear solvers?, Frontiers in big data 4 (2021).

[3] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, The Journal of Machine Learning Research 13 (2012) 281–305.

[4] A. Abraham, Meta-learning evolutionary artificial neural networks, CoRR cs.AI/0405024 (2004).

[5] P. Ren, Y. Xiao, X. Chang, P. Huang, Z. Li, X. Chen, X. Wang, A comprehensive survey of neural architecture search: Challenges and solutions, ACM Comput. Surv. 54 (4) (2021).

[6] B. Zoph, Q. Le, Neural architecture search with reinforcement learning, in: International Conference on Learning Representations, 2017.

[7] B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning, in: International Conference on Learning Representations, 2017.

[8] I. Bello, B. Zoph, V. Vasudevan, Q. V. Le, Neural optimizer search with reinforcement learning, in: International Conference on Machine Learning, PMLR, 2017, pp. 459–468.

[9] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, A. Kurakin, Large-scale evolution of image classifiers, in: Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, Vol. 70, 2017, pp. 2902–2911.

[10] X. Chen, L. Xie, J. Wu, Q. Tian, Progressive differentiable architecture search: Bridging the depth gap between search and evaluation, in: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), IEEE Computer Society, 2019, pp. 1294–1303.

[11] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, K. Murphy, Progressive neural architecture search, in: Proceedings of the European conference on computer vision (ECCV), 2018, pp. 19–34.

[12] F. Assunção, N. Lourenço, P. Machado, B. Ribeiro, Automatic generation of neural networks with structured grammatical evolution, Congress on Evolutionary Computation 392 (2017) 1557–1564.

[13] P. P. Palmes, T. Hayasaka, S. Usui, Mutation-based genetic neural network, Transactions on Neural Networks 16 (2005) 587–600.

[14] S. Zhang, H. Wang, L. Liu, C. Du, J. Lu, Optimization of neural network based on genetic algorithm and bp, Proceedings of 2014 International Conference on Cloud Computing and Internet of Things (2014) 203–207.

[15] J.-T. Tsai, J.-H. Chou, T.-K. Liu, Tuning the structure and parameters of a neural network by using hybrid taguchi-genetic algorithm, Transactions on Neural Networks 17 (2006) 69–80.

[16] P. Vidnerová, R. Neruda, Evolution strategies for deep neural network models design, in: Conference on Theory and Practice of Information Technologies, 2017, p. 159–166.

[17] P. Escapil-Inchauspé, G. A. Ruz, Hyper-parameter tuning of physics-informed neural networks: Application to helmholtz problems, ArXiv abs/2205.06704 (2022).

[18] X. Chen, J. Duan, G. E. Karniadakis, Learning and meta-learning of stochastic advection-diffusion-reaction systems from sparse measurements, Physica A. (2019).

[19] E. Skomski, J. Drgoňa, A. Tuor, Physics-informed neural state space models via learning and evolution, in: Proceedings of the 3rd Conference on Learning for Dynamics and Control, PMLR 144, Vol. 144, 2021, pp. 980–991.

[20] W. Yicheng, H. Xiaotian, C. Chia-Yuan, Z. Daochen, B.-N. Ulisses, H. Xia, Auto-pinn: Understanding and optimizing physics-informed neural architecture, ArXiv abs/2205.13748 (2022).

[21] Y. Wang, L. Zhong, Nas-pinn: Neural architecture search-guided physics-informed neural network for solving pdes, arXiv preprint arXiv:2305.10127 (2023).

[22] J. Bergstra, D. Yamins, D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in: International conference on machine learning, PMLR, 2013, pp. 115–123.

[23] M. Raissi, P. Perdikaris, G. 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 (2019) 686–707.

[24] D. C. Psichogios, L. H. Ungar, A hybrid neural network-first principles approach to process modeling, AIChE Journal 38 (1992) 1499–1511.

[25] I. Lagaris, A. Likas, D. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Transactions on Neural Networks 9 (1998) 987–1000.

[26] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, Tensorflow: A system for large-scale machine learning, in: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, 2016, p. 265–283.

[27] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in pytorch, in: 31st Conference on Neural Information Processing Systems, 2017.

[28] M. Melanie, An Intoduction to Genetic Algorithms, MIT press, 1998.

[29] B. Stojanovic, M. Milivojevic, N. Milivojevic, D. Antonijevic, A self-tuning system for dam behavior modeling based on evolving artificial neural networks, Advances in Engineering Software 97 (2016) 85–95.

[30] A. Auger, J. Bader, D. Brockhoff, E. Zitzler, Theory of the hypervolume indicator: Optimal μ-distributions and the choice of the reference point, in: Proceedings of the Tenth ACM SIGEVO Workshop on Foundations of Genetic Algorithms, 2009, p. 87–102.

[31] B. Stojanovic, M. Milivojevic, M. Ivanovic, N. Milivojevic, D. Divac, Adaptive system for dam behavior modeling based on linear regression and genetic algorithms, Advances in Engineering Software 65 (2013) 182–190.

[32] M. Ivanovic, V. Simic, B. Stojanovic, A. Kaplarevic-Malisic, B. Marovic, Elastic grid resource provisioning with wobingo: A parallel framework for genetic algorithm based optimization, Future Generation Computer Systems (2014).

[33] V. Simic, B. Stojanovic, M. Ivanovic, Optimizing the performance of optimization in the cloud environment–an intelligent auto-scaling approach, Future Generation Computer Systems (2019).

[34] M. Ivanovic, V. Simic, Efficient evolutionary optimization using predictive auto-scaling in containerized environment, Applied Soft Computing 129 (2022) 109610.

[35] M. Ivanovic, M. Svicevic, S. Savovic, Numerical solution of stefan problem with variable space grid method based on mixed finite element/finite difference approach, International Journal of Numerical Methods for Heat and Fluid Flow (2017).

[36] S. Savovic, J. Caldwell, Numerical solution of stefan problem with time-dependent boundary conditions by variable space grid method, Thermal Science 13 (2009) 165–174.

[37] F. Ihlenburg, Finite element analysis of acoustic scattering, Springer, 1998.

[38] E. Haghighat, R. Juanes, Sciann: A keras wrapper for scientific computations and physics-informed deep learning using artificial neural networks, CoRR abs/2005.08803 (2020).

[39] L. Lu, X. Meng, Z. Mao, G. E. Karniadakis, Deepxde: A deep learning library for solving differential equations, CoRR abs/1907.04502 (2019).