Can Transfer Neuroevolution Tractably Solve Your Differential Equations?

Jian Cheng Wong, Abhishek Gupta, and Yew-Soon Ong, *Fellow, IEEE*

***Abstract*— This paper introduces neuroevolution for solving differential equations. The solution is obtained through optimizing a deep neural network whose loss function is defined by the residual terms from the differential equations. Recent studies have focused on learning such *physics-informed neural networks* through stochastic gradient descent (SGD) variants, yet they face the difficulty of obtaining an accurate solution due to optimization challenges. In the context of solving differential equations, we are faced with the problem of finding globally optimum parameters of the network, instead of being concerned with out-of-sample generalization. SGD, which searches along a single gradient direction, is prone to become trapped in local optima, so it may not be the best approach here. In contrast, neuroevolution carries out a parallel exploration of diverse solutions with the goal of circumventing local optima. It could potentially find more accurate solutions with better optimized neural networks. However, neuroevolution can be slow, raising tractability issues in practice. With that in mind, a novel and computationally efficient *transfer neuroevolution* algorithm is proposed in this paper. Our method is capable of exploiting relevant experiential priors when solving a new problem, with adaptation to protect against the risk of negative transfer. The algorithm is applied on a variety of differential equations to empirically demonstrate that transfer neuroevolution can indeed achieve better accuracy and faster convergence than SGD. The experimental outcomes thus establish transfer neuroevolution as a noteworthy approach for solving differential equations, one that has never been studied in the past. Our work expands the resource of available algorithms for optimizing physics-informed neural networks.**

***Index Terms*—Transfer neuroevolution, differential equations, physics-informed neural networks.**

I. Introduction

SOLVING ordinary differential equations (ODEs) and partial differential equations (PDEs) is the cornerstone of scientific modelling in modern science and engineering [1-3]. The solution of these differential equations allows us to understand and make predictions on the behaviors of physical systems in a wide variety of scientific problems, including heat and mass transfer, optics, acoustics, material elasticity, electromagnetic waves, fluid dynamics, and many other dynamical processes in physics [4, 5], sociology [6], finance and economics [7], etc. The idea of using neural networks to solve differential equations goes back to early works in the 1990s [8-14]. Given the surging popularity of deep learning lately and advancement in computing capability, there has been renewed interest in designing deep neural networks (DNNs) to solve differential equations [15-23]. In brief, a DNN – also termed herein as a physics-informed neural network as per Raissi *et al.* [24] – is constructed such that its output $\hat{u}(x,t)$ generates the solution u of the governing differential equations in space $x\epsilon\Omega$ and time $t\epsilon[0,T]$ domains. The loss function of such a DNN is defined by the residual terms from the differential equations, under prescribed initial and boundary conditions. In principle, *infinite* training data instances are accessible by sampling arbitrary points within a problem's spatial-temporal domain. The loss thus acts as a penalty to constrain the DNN from violating the governing equations at *all* points. *In other words, the problem of solving differential equations is transformed to one of global optimization, in which the loss is to be minimized to zero.*

Different from classical numerical solvers [25], the DNN approach has the main advantage of being mesh-free. Meshing (i.e., spatial discretization) itself is a nontrivial task. Classical numerical schemes are prone to failure in finding the right solution if the meshing is not appropriately done. Their accuracy is limited by the size of the discretization and interpolation scheme used (usually linear). Being universal approximators, DNNs offer superior approximation or even exact replication to the solution. Moreover, the solution via DNN is differentiable. Its relatively compact representation requires lower memory demand for storage. Beyond solving differential equations (referred to as the forward problem), physics-informed neural networks can be further extended to solve inverse problems such as designing metamaterials [26], inferring unknown dynamic from observations [27, 28], and quantifying fluid flows from visualizations or sensors data [29-31]; see examples in Fig. 1. They offer a path to *rationalizable* artificial and computational intelligence systems that are consistent with fundamental physics laws [32]. These advantages brought by synergizing DNNs and differential equations are however attainable at the cost of a steep

Jian Cheng Wong is with the Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), Singapore, and is also with the School of Computer Science and Engineering, NTU, Singapore (email: wongj@ihpc.a-star.edu.sg).

Abhishek Gupta is with the Singapore Institute of Manufacturing Technology (SIMTech), Agency for Science, Technology and Research (A*STAR), Singapore (email: abhishek_gupta@simtech.a-star.edu.sg).

Yew-Soon Ong is Chief Artificial Intelligence Scientist with the Agency for Science, Technology and Research (A*STAR), Singapore, and is also with the Data Science and Artificial Intelligence Research Centre, School of Computer Science and Engineering, NTU, Singapore (email: asysong@ntu.edu.sg).

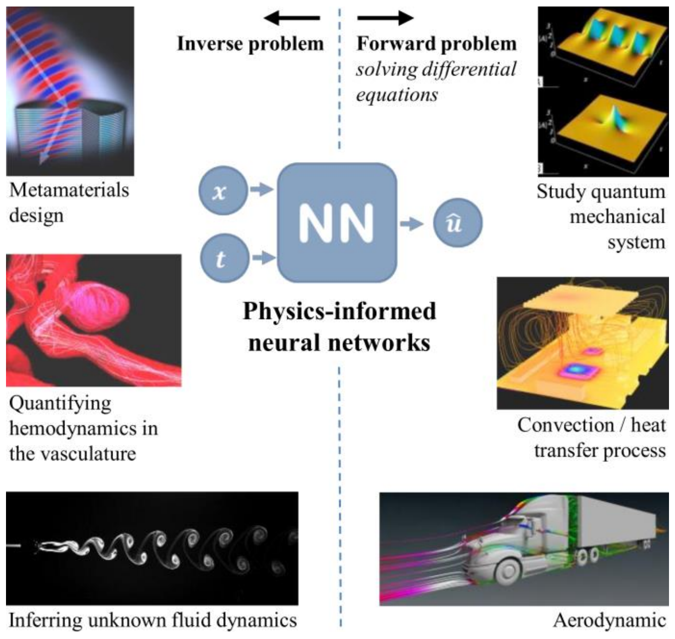

Inverse problem
Forward problem
solving differential equations
Metamaterials design
Study quantum mechanical system
x
t
NN
û
Physics-informed neural networks
Quantifying hemodynamics in the vasculature
Convection / heat transfer process
Inferring unknown fluid dynamics
Aerodynamic

Fig. 1. Applications of physics-informed neural networks. By solving differential equations, accurate predictions on the behaviors of physical systems (for example, quantum mechanics, heat and mass transfer, fluid dynamics) can be made in a wide variety of problems across scientific fields. DNNs can also be extended to solve inverse problems, such as designing metamaterials, quantifying hemodynamics in the vasculature from passive scalar, inferring unknown physical quantities, etc.

optimization challenge caused by high-dimensional, non-convex loss function landscapes.

When considering the optimization of physics-informed neural networks, there has been a lack of research investigation beyond mainstream gradient descent methods, such as stochastic gradient descent (SGD) and Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithms. Their known tendency of becoming trapped in local minima could nevertheless hinder a DNN from approximating the right solution. Although SGD has been very successful in deep learning, the emphasis is typically to learn a model from noisy data with good out-of-sample generalization. Over-fitting to the training data is discouraged, and a local minimum often suffices in achieving high test accuracy. However, the requirement is somewhat different when it comes to solving differential equations, in which globally optimum model parameters are sought. Generalizability and over-fitting are not a concern given access to potentially infinite training data covering the entire problem domain; rather, having a pre-maturely converged physics-informed neural network implies an unphysical solution. For these reasons, it is contended that SGD may not necessarily be the best approach for optimization in this domain.

An alternative approach comes from the field of *neuroevolution* [33], which uses evolutionary algorithms (EAs) to optimize DNNs. Their main conceptual distinction from SGD is that SGD follows a single gradient direction, whereas EAs search with a population of diverse solutions with the goal of circumventing local optima [34]. This makes neuroevolution a different paradigm from gradient descent, since the notion of diversity does not explicitly exist in the latter [35]. As such, neuroevolution offers a promising substitute to SGD for global optimization [36]; such as for solving differential equations. In particular, it is demonstrated in this paper that neuroevolution via a state-of-the-art EA, namely *natural evolution strategies* (NES) [37], outperforms SGD in solving a variety of differential equation, providing physically accurate solutions. Neuroevolution is thus highlighted as a noteworthy approach for solving differential equations.

Neuroevolution can however be slow to converge compared to gradient descent. To enhance computational tractability, we propose a novel augmentation of neuroevolution with *transfer optimization* [38-40], where information in the form of *experiential priors* are reused from past (*source*) problem instances to boost the *target* search. In a scientific study, it is common for a single set of differential equations to be evaluated under different environments and boundary conditions. Relevant experiences are therefore naturally accumulated over time, and can be exploited given any new target problem. To this end, we develop a new transfer optimization method which can be integrated with probabilistic model-based evolution strategies, such as NES (or even others like CMA-ES or OpenAI-ES [41, 42]). Different from the commonly used transfer strategy of fixing pre-trained neural network layers [43, 44], our method features a probability mixture model-based adaptive design to protect the algorithm from the risk of negative transfer. The mixture model allows the joint processing of multiple sources, adaptively selecting the one that is most relevant to the target task. In the experimental study, the proposed transfer optimization algorithm is integrated with the NES, empirically demonstrating improvements in convergence speed and accuracy over baseline neuroevolution and SGD.

The remainder of the paper is organized as follows. In the next section, physics-informed neural networks for differential equation problem is described. In Section III, the methodology of neuroevolution via probabilistic model-based evolution strategies is introduced. The proposed method for achieving transfer neuroevolution is then presented in Section IV. Section V presents the experimental study to illustrate the competitive advantage of transfer neuroevolution over SGD across several test problems. Finally, Section VI contains the conclusion and directions for future research.

II. Problem Setup: Physics-Informed Neural Networks for Differential Equations

A. Differential Equations

Consider differential equations of the general form:

$$u_t(x,t) + \mathcal{N}_x[u(x,t)] = 0, \;\; x\epsilon\Omega, t\epsilon[0,T], \tag{1}$$

$$u(x,0) = u_o(x), \;\; x\epsilon\Omega, \tag{1b}$$

$$\mathcal{B}[u(x,t)] = g(x,t), \;\; x\epsilon\partial\Omega, t\epsilon[0,T], \tag{1c}$$

where $u_t(x,t)$ is the temporal derivative, and $\mathcal{N}_x[\cdot]$ is a general nonlinear differential operator which includes non-linear terms of spatial derivatives, such as the first and second order derivatives $u_x(x,t)$ and $u_{xx}(x,t)$, respectively. The differential equation (1) usually describes certain dynamical process in the

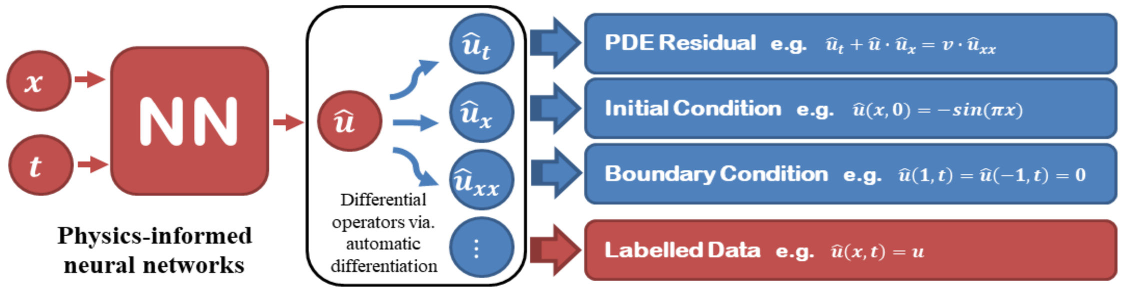

x
t
NN
$\hat{u}$
$\hat{u}_t$
$\hat{u}_x$
$\hat{u}_{xx}$
⋮
Differential operators via. automatic differentiation
Physics-informed neural networks
PDE Residual e.g. $\hat{u}_t + \hat{u}\cdot\hat{u}_x = v\cdot\hat{u}_{xx}$
Initial Condition e.g. $\hat{u}(x,0) = -sin(\pi x)$
Boundary Condition e.g. $\hat{u}(1,t) = \hat{u}(-1,t) = 0$
Labelled Data e.g. $\hat{u}(x,t) = u$

Fig. 2. Physics-informed neural networks consist of additional residual (loss) terms from the differential equation, initial and boundary conditions (blue), which require the computation of differential operators of the neural network output with respect to the inputs. Such physics-informed neural networks can emulate the solution of differential equations without the need for labelled data. We have a global optimization problem (instead of a learning problem) to find the most physically accurate solution to the target differential equation.

physical world. The corresponding spatial domain can be of 1-, 2- or 3-dimensions. As an example, the Burgers' equation reads $u_t + u\cdot u_x - \nu\cdot u_{xx} = 0$, by having $\mathcal{N}_x[u] = u\cdot u_x - \nu\cdot u_{xx}$. The nonlinear differential operator in Burgers' equation describes the advection and diffusion processes of the physical quantity u in 1D, and contains a problem specific diffusion coefficient ν.

The interest of this paper lies in finding the solution $u(x,t)$ which satisfies the differential equation (1) across space $x\epsilon\Omega$ and time $t\epsilon[0,T]$ domains. Such solution may not be unique unless sufficient initial condition (1b) and/or boundary condition (1c) are given. The initial condition at $t = 0$ is defined by $u_o(x)$, and the boundary operator $\mathcal{B}[\cdot]$ enforces the desired condition $g(x,t)$ at the domain boundary $\partial\Omega$. This $\mathcal{B}[\cdot]$ can be an identity operator (Dirichlet boundary condition) or a differential operator (Neumann boundary condition). A meaningful solution would need to satisfy all these conditions, in addition to the main differential equation.

B. *Physics-Informed Neural Networks*

To solve for the differential equation in (1), the neural network approach constructs a DNN representation $\hat{u}(x,t;\boldsymbol{w})$ to emulate the unknown solution u, with network parameters $\boldsymbol{w} = \{w_i\}_{i=1}^{d}$ to be optimized. In the present study, the network architecture and other hyper-parameters are specified, making $\boldsymbol{w}$ refer to the weights of the neural network. A popular term for such neural network is "physics-informed neural network", because they use the residual terms from the differential equation (1), and the prescribed initial (1b) and boundary (1c) conditions as the loss function, which is defined below:

$$\mathcal{L} = \mathcal{L}_{DE} + \beta_{IC}\cdot\mathcal{L}_{IC} + \beta_{BC}\cdot\mathcal{L}_{BC}, \tag{2}$$

where,

$$\mathcal{L}_{DE} = \|\hat{u}_t(\cdot\,;\boldsymbol{w}) + \mathcal{N}_x[\hat{u}(\cdot\,;\boldsymbol{w})]\|^2_{2,\Omega\times[0,T]}, \tag{2b}$$
$$\mathcal{L}_{IC} = \|\hat{u}(\cdot\,,0\,;\boldsymbol{w}) - u_0\|^2_{2,\Omega}, \tag{2c}$$
$$\mathcal{L}_{BC} = \|\mathcal{B}[\hat{u}(\cdot\,;\boldsymbol{w})] - g(\cdot)\|^2_{2,\partial\Omega\times[0,T]}. \tag{2d}$$

The relative weights βs in (2) control the trade-off between different terms in the loss function, and may need to be scaled in a problem-specific way. Physics-informed neural networks may also include conventional data loss (for example, when solving an inverse problem). In the case of solving ODEs and PDEs, the loss function only comprises of the residual terms with respect to the differential equation, initial and boundary conditions; see illustration in Fig. 2.

C. *Computation of the Loss*

The computation of the loss (2) involves substitution of the DNN output $\hat{u}$ into the differential equation for evaluating the residuals (2b) over the computational domain, as well as matching the output $\hat{u}$ against initial condition (2c) at $t = 0$, and boundary conditions (2d) over the domain boundary $\partial\Omega$. For certain types of activation functions (for example sigmoid, softplus, tanh, etc.), the DNN output is higher order differentiable with respect to its spatial and temporal inputs. These differential operators, such as $\hat{u}_t(x,t;\boldsymbol{w}), \hat{u}_x(x,t;\boldsymbol{w})$, $\hat{u}_{xx}(x,t;\boldsymbol{w})$, are required for the evaluation of the loss. They can be conveniently obtained via automatic differentiation [45].

Although the residual terms (2b-d) are defined over a continuous computational domain, for practical reasons, we can only compute the mean squared residuals of a candidate solution over a finite set of m spatial-temporal collocation points $\mathcal{D} = \{(x_i,t_i)\}_{i=1}^{m}$. These points are sampled (for example, using randomized Latin hypercube sampling) from the respective computational domains. In this paper paper, a dynamic sampling scheme is adopted such that for every loss evaluation a new batch of m collocation points are generated.

The evaluated loss (2) indicates how well the DNN output emulates the true solution of the differential equation. Ideally, this loss has a global minimum of 0, given that the output exactly satisfies the governing equations over the entire computational domain. In this regard, we are faced with a global optimization problem (instead of a learning generalization problem) for finding the most physically accurate solution to the target differential equation.

III. Methodology: Neuroevolution

The central thesis of this paper is the untapped efficacy of evolutionary algorithms for optimizing the DNN in generating good solutions to the differential equations. In particular, the objective is to find the best d-dimensional neural network weights $\boldsymbol{w} \in R^d$ which minimizes the loss (2). As opposed to typical gradient descent methods such as SGD, which searches along a single gradient direction, EAs search with a population of diverse solutions $\{\boldsymbol{w}_k\}_{k=1}^{\lambda}$ for better *fitness*. Here, λ is the population size and the fitness $f = -\mathcal{L}$ refers to the negative loss of the physics-informed neural network, which is evaluated on a dynamically sampled batch of m collocation points. The diversity in an EA population could potentially be the key factor to overcome local optima and thus achieve better optimized neural networks.

Among EAs, there is a subgroup of techniques such as NES, CMA-ES, and OpenAI-ES, that adopt a probabilistic model, called the *search distribution*, to represent the population. In a nutshell, they keep tracing an evolving search distribution and produce pseudo-offspring by drawing new samples from the distribution. As the search progresses, the distribution is iteratively updated towards regions with higher population fitness. All probabilistic model-based evolution strategies follow this basic principle, although they may differ in their distribution update mechanism. They have shown to be an effective method to evolve neural network weights $\boldsymbol{w}$ in continuous domains, by using a d-dimensional multivariate normal search distribution [37, 42, 46-49].

For demonstration purposes, we consider the state-of-the-art NES as the baseline neuroevolution algorithm in this paper. The underlying objective function of NES can be expressed as:

$$\mathcal{J}(\boldsymbol{\theta}) = \mathrm{E}_{\boldsymbol{\theta}}[f(\boldsymbol{w})] = \int f(\boldsymbol{w})\, \pi(\boldsymbol{w}|\boldsymbol{\theta})\, d\boldsymbol{w}, \tag{3}$$

which is the expected fitness of the population represented by probabilistic model $\pi(\boldsymbol{w}|\boldsymbol{\theta})$, where the search distribution is specified by *distributional parameters* $\boldsymbol{\theta}$. In this paper, the d-dimensional multivariate normal search distribution is parameterized by $\boldsymbol{\theta} = (\mu, A)$, where $A\,A^T = \Sigma$. The distribution mean $\mu \in R^d$ and full covariance matrix $\Sigma \in R^{d\times d}$ characterize the search center and mutation. A candidate solution for the neural network weights is therefore a realization of the normally distributed random variable $\boldsymbol{w} \sim N(\mu, \Sigma)$.

It's worth noting that NES searches in the space of distributional parameters, but not the problem space. In contrast, the objective function for typical gradient descent methods such as SGD is to search for an optimal $\boldsymbol{w}$ directly; i.e., $\mathcal{J}(\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w})$. This further highlights the conceptual distinction between the two approaches.

In particular, NES utilizes the *search gradient* $\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta})$ on the expected fitness (3) to evolve the search distribution. The gradient can be estimated from a population of *pseudo-offspring* drawn from the current search distribution as [37]:

$$\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}) \approx \frac{1}{\lambda}\sum_{k=1}^{\lambda} f(\boldsymbol{w}_k)\, \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{w}_k|\boldsymbol{\theta})), \tag{4}$$

where λ is the population size and $f(\boldsymbol{w}_k)$ is the fitness of the k^{th} sampled offspring. Then, the distributional parameters are updated based on the estimated search gradient as:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \cdot \mathrm{F}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}), \tag{5}$$

where η is the learning rate. In (5) the search gradient is normalized by the inverse of the *Fisher information matrix* $\mathrm{F} = \mathrm{E}_{\boldsymbol{\theta}}[\nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{w}|\boldsymbol{\theta})\ \nabla_{\boldsymbol{\theta}} \log \pi(\boldsymbol{w}|\boldsymbol{\theta})^{\mathrm{T}}]$ (i.e., the variance of the gradient) for the given distributional parameters to form the *natural gradient* $\widetilde{\nabla}_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}) = \mathrm{F}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta})$, which is the hallmark of NES. This natural gradient takes into consideration the uncertainty of gradient estimates when providing an ascent direction in the space of distributional parameters.

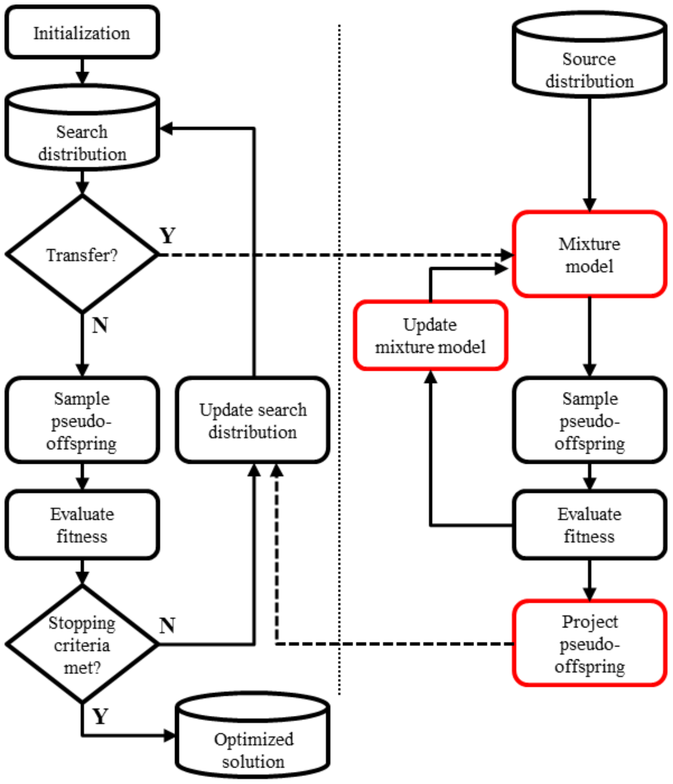


Fig. 3. A conceptual illustration of the proposed mixture model-based adaptive transfer, featuring the methodology (1) to construct and update a mixture model of the target search and source distributions, and (2) to influence the search distribution update on the target problem based on the projection of pseudo-offspring sampled from the source distributions (boxes highlighted in red). The proposed transfer method is suitable for general probabilistic model-based evolution strategies as shown on the left side of the vertical dotted line. The interface between the baseline evolution strategies and additional transfer components are indicated by dashed arrows.

IV. Transfer Neuroevolution

In this section, we augment neuroevolution with transfer optimization, boosting search efficiency by reusing experiential priors from related source problem instances. This capability is enabled via a mixture model-based adaptive transfer method. The proposed method is suitable for general probabilistic model-based evolutionary strategies. It is worth noting that the approach makes no strict assumption on the synergy between source and target problems. The adaptive transfer method features a dynamic mechanism to automatically exploit useful experience from source problems, such that better quality pseudo-offspring can be induced to effectively influence the target search distribution update. Importantly, the adaptation mechanism is able to retreat from irrelevant sources, to curb negative transfer. Fig. 3 gives the conceptual illustration for the proposed mixture model-based adaptive transfer method.

A. *Transfer via Mixture Modelling*

Let us consider a search distribution $\pi(\boldsymbol{w}|\boldsymbol{\theta})$ for the target optimization problem, and a single source distribution $\varphi(\boldsymbol{w})$ for simplicity of exposition. It is assumed that the source is expressed in the form of a distributional prior, for example, a search distribution acquired from successful evolution of the same neural network with NES (or any other probabilistic model-based evolution strategies like CMA-ES or OpenAI-ES) for solving a similar differential equations problem. To facilitate knowledge transfer, a mixture model to unify the target and source distributions is defined as:

$$m(\boldsymbol{w}) = \alpha_\pi\, \pi(\boldsymbol{w}) + \alpha_\varphi\, \varphi(\boldsymbol{w}), \tag{6}$$

where α_π and α_φ are the mixing coefficients for the target and source distribution components, respectively. This mixture model can be initiated with arbitrary mixing coefficients, subject to the constraint $\alpha_\pi + \alpha_\varphi = 1$. In each iteration of neuroevolution, a predefined parameter is used to ascertain if the knowledge transfer mode is to be *activated*. In the transfer mode, λ pseudo-offspring are sampled from the mixture model $m(\boldsymbol{w})$ instead of the search distribution $\pi(\boldsymbol{w}|\boldsymbol{\theta})$, in effect inducing pseudo-offspring from the source to the target problem. Otherwise, the algorithm continues to draw pseudo-offspring from the target search distribution.

It is important to note that the target search distribution is *evolving* (see Section IV-C), while the source distribution is *fixed* during the optimization process. The key step is how to dynamically update the mixing coefficients α_π and α_φ when the search progresses, where α_φ can be viewed as the degree of transfer from source to target. If the source problem is beneficial, we would want to increase α_φ so that more high-quality pseudo-offspring can be sampled from the source distribution to influence the target search. However, after a certain point, the pseudo-offspring from the source may no longer remain competitive with those sampled from the evolved target search distribution. At that point, we would want to gradually reduce α_φ, eventually deactivating the source distribution from the mixture model, i.e., setting $\alpha_\varphi = 0$.

Decidedly, how beneficial a pseudo-offspring is to the target problem shall be reflected by its fitness. To this end, we propose to update the mixing coefficients towards better *expected fitness* under the mixture model, with the following generalization of (3):

$$\mathcal{J}^m = \int f(\boldsymbol{w})\,\left[\alpha_\pi\, \pi(\boldsymbol{w}) + \alpha_\varphi\, \varphi(\boldsymbol{w})\right] d\boldsymbol{w}. \tag{7}$$

Then, the gradient estimates for α_π and α_φ on the expected fitness can be derived, using the log-likelihood trick, as:

$$\begin{aligned}\nabla_{\alpha_\pi}\mathcal{J}^m &= \textstyle\sum_{k=1}^{\lambda} f(\boldsymbol{w}_k)\, \nabla_{\alpha_\pi} \log\left(\alpha_\pi\, \pi(\boldsymbol{w}_k) + \alpha_\varphi\, \varphi(\boldsymbol{w}_k)\right)\\ &= \textstyle\sum_{k=1}^{\lambda} f(\boldsymbol{w}_k)\, \frac{\pi(\boldsymbol{w}_k)}{\alpha_\pi\, \pi(\boldsymbol{w}_k) + \alpha_\varphi\, \varphi(\boldsymbol{w}_k)},\end{aligned} \tag{8a}$$

and,

$$\nabla_{\alpha_\varphi}\mathcal{J}^m = \textstyle\sum_{k=1}^{\lambda} f(\boldsymbol{w}_k)\, \frac{\varphi(\boldsymbol{w}_k)}{\alpha_\pi\, \pi(\boldsymbol{w}_k) + \alpha_\varphi\, \varphi(\boldsymbol{w}_k)}, \tag{8b}$$

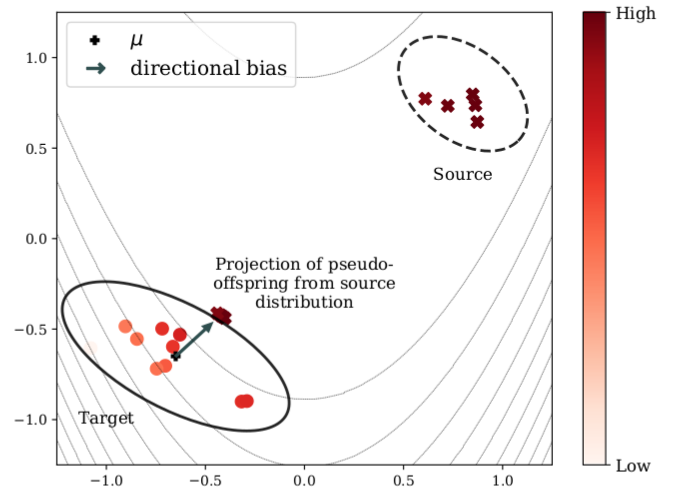


Fig. 4. An illustration of projecting the pseudo-offspring (marker: cross) produced by the source distribution towards the target search distribution. The projected pseudo-offspring are within r radial distance of the search distribution center, while still preserving the same directional and fitness information (indicated by color scale). By combining them with the pseudo-offspring originating from the target distribution (marker: round), a definite directional bias is induced on the evolution of the target search without absurdly changing it in a single update step; thus avoiding numerical instability issues.

given λ pseudo-offspring $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_\lambda$ sampled from the mixture model and their fitness $f(\boldsymbol{w}_1), f(\boldsymbol{w}_2), \ldots, f(\boldsymbol{w}_\lambda)$. With these gradient estimates, the mixing coefficients can be updated as follows:

$$\alpha_\pi \leftarrow \alpha_\pi + \eta_\alpha \cdot \nabla_{\alpha_\pi}\mathcal{J}^m, \tag{9a}$$
$$\alpha_\varphi \leftarrow \alpha_\varphi + \eta_\alpha \cdot \nabla_{\alpha_\varphi}\mathcal{J}^m, \tag{9b}$$

where η_α is the learning rate. Note that the constraint $\alpha_\pi + \alpha_\varphi = 1$ can be easily imposed by normalization. The mixture model formulation described above can be seamlessly extended to multiple components for multiple sources transfer, making the method more powerful since the chance of having useful experience included increases with the number of sources.

B. *Influencing Evolution of the Target Search Distribution*

After sampling pseudo-offspring from the mixture model and evaluating their fitness, the target search distribution is to be updated. Broadly speaking, we expect the target distribution to gradually evolve with the guidance of pseudo-offspring transferred from the source. In this way, it has greater chance to explore diverse and high-quality solutions along the search path. However, absurdly large changes to the mean μ of the target search distribution in a single update step should be avoided. At the same time, it is not advisable to be overly-greedy in expanding the covariance matrix Σ when good pseudo-offspring (from the source) are very far away from the target distribution. A direct induction of pseudo-offspring from the source may thus result in such deleterious outcomes, often leading to numerical instability issues.

To understand this issue, let us assume a pseudo-offspring $\boldsymbol{w}_k$ with fitness $f(\boldsymbol{w}_k)$ is being induced from the source distribution. Its contribution to the distributional parameter gradient estimates are $f(\boldsymbol{w}_k)\boldsymbol{z}_k$ and $f(\boldsymbol{w}_k)(\boldsymbol{z}_k\boldsymbol{z}_k{}^T - \mathbb{I})$ [49, 50], where $\boldsymbol{z}_k = A^{-1}(\boldsymbol{w}_k - \mu)$ maps $\boldsymbol{w}_k$ into the target search distribution's natural coordinates; μ and A are the center and the square root of the covariance matrix of the target search distribution, respectively. If $\boldsymbol{w}_k$ is far from μ in relation to A, then the value $\boldsymbol{z}_k = A^{-1}(\boldsymbol{w}_k - \mu)$ will be extremely large. As a result, $\boldsymbol{z}_k\boldsymbol{z}_k{}^T$ will be further exaggerated. This could cause the gradient estimates to (numerically) explode, causing failure of the distributional parameter updates. Such instability is likely to occur even in moderate dimensions, due to distribution sparsity as a consequence of the curse of dimensionality.

To overcome the instability issue, we propose to project the source distribution's pseudo-offspring closer to the target search distribution before updating the distributional parameters. While doing so however, the original fitness values of the projected offspring are retained. A simple strategy to perform such projection, under the assumption that the target follows a multivariate normal distribution – a common practice in most probabilistic model-based evolution strategies – is outlined below. First, we define a tunable parameter r to represent a standardized radial distance from the distribution center. Choosing say r=3 implies that only those pseudo-offspring that are more than 3 standard deviations away from the target distribution center are projected. In particular, the direction vector $d_k = \boldsymbol{w}_k - \mu$ is computed, and then the corresponding pseudo-offspring $\boldsymbol{w}_k$ is projected to $\widetilde{\boldsymbol{w}}_k$ as shown below (so that it lies within radial distance r of the center):

$$\boldsymbol{w}_k \to \widetilde{\boldsymbol{w}}_k = \mu + d_k \times \min\left(1, \frac{r}{\|A^{-1}d_k\|}\right). \qquad (10)$$

As demonstrated in Fig. 4, these projected pseudo-offspring are now closer to the target distribution, while preserving directional and fitness information for the update. As a result, they induce a definite directional bias on the evolution of the target search distribution without absurdly changing it in a single update.

Our formulation of transfer neuroevolution with mixture modelling and pseudo-offspring projection is generic for probabilistic model-based search, and does not depend on a specific algorithm. The next section presents a particular instantiation of this method by implementing it on a variant of NES. This transfer NES algorithm will be subsequently applied to solve differential equations problems.

C. An Instantiation: Transfer NES (tNES) Algorithm

In this section, transfer neuroevolution with a variant of NES, namely the *exponential NES* (xNES) algorithm [50] is presented. The xNES efficiently updates the mean μ and the full covariance matrix $\Sigma = A\,A^T$ of a search distribution via a coordinate transformation trick, to avoid trivial operations on computing and inverting the Fisher information matrix in (5). The details of the xNES algorithm can be found in [50]. Algorithm 1 presents the pseudocode of our transfer implementation on xNES, which is referred to hereafter as *t*NES.

Algorithm 1: Pseudocode of *t*NES

```
Input: optimization problem $f \in R^d \to R$, initial search distribution $\mu \in R^d$, $A = \sigma\mathbb{I} \in R^{d\times d}$, source distribution $\mu_\varphi \in R^d$, $A_\varphi \in R^{d\times d}$, initial mixture coefficient $\alpha_\varphi \in [0,1]$
$\alpha_\pi \leftarrow 1 - \alpha_\varphi$
while stopping criteria not met do
    $p \leftarrow \alpha_\varphi$ if transfer activated else $p \leftarrow 0$
    for $k \in \{1, 2, \ldots, \lambda\}$ do
        sample $\boldsymbol{z}_k \sim N(0, \mathbb{I})$, $\boldsymbol{s}_k \sim bernoulli(p)$
        if $\boldsymbol{s}_k = 1$ then
            $\boldsymbol{w}_k \leftarrow A_\varphi \boldsymbol{z}_k + \mu_\varphi$
            $d_k \leftarrow \boldsymbol{w}_k - \mu$
            $\widetilde{\boldsymbol{w}}_k \leftarrow \mu + d_k \times \min\left(1, \frac{r}{\|A^{-1}d_k\|}\right)$
            $\boldsymbol{z}_k \leftarrow A^{-1}(\widetilde{\boldsymbol{w}}_k - \mu)$
        else
            $\boldsymbol{w}_k \leftarrow A\boldsymbol{z}_k + \mu$
        end
    end
    sort $\{(\boldsymbol{z}_k, \boldsymbol{w}_k)\}$ w.r.t. $f(\boldsymbol{w}_k)$
    compute $\{u_k\}$
    $\nabla_\mu \mathcal{J} \leftarrow \sum_{k=1}^{\lambda} u_k\, \boldsymbol{z}_k$
    $\nabla_A \mathcal{J} \leftarrow \sum_{k=1}^{\lambda} u_k\, (\boldsymbol{z}_k\boldsymbol{z}_k{}^T - \mathbb{I})$
    $\mu \leftarrow \mu + \eta_\mu \cdot \nabla_\mu \mathcal{J}$
    $A \leftarrow A \cdot \exp(1/2 \cdot \eta_A \cdot \nabla_A \mathcal{J})$
    if transfer activated then
        $\nabla_{\alpha_\pi}\mathcal{J}^m = \sum_{k=1}^{\lambda} u_k \frac{\pi(\boldsymbol{w}_k)}{\alpha_\pi\,\pi(\boldsymbol{w}_k) + \alpha_\varphi\,\varphi(\boldsymbol{w}_k)}$
        $\nabla_{\alpha_\varphi}\mathcal{J}^m = \sum_{k=1}^{\lambda} u_k \frac{\varphi(\boldsymbol{w}_k)}{\alpha_\pi\,\pi(\boldsymbol{w}_k) + \alpha_\varphi\,\varphi(\boldsymbol{w}_k)}$
        $\alpha_\pi \leftarrow \alpha_\pi + \eta_\alpha \cdot \nabla_{\alpha_\pi}\mathcal{J}^m$
        $\alpha_\varphi \leftarrow \alpha_\varphi + \eta_\alpha \cdot \nabla_{\alpha_\varphi}\mathcal{J}^m$
        normalize $(\alpha_\pi, \alpha_\varphi)$ s.t. $\alpha_\pi + \alpha_\varphi = 1$
    end
end
```

TABLE I: TUNING PARAMETERS OF tNES

$\{\Delta t, t_{max}\}$: transfer plan
λ : population size
r: standardize radial distance for pseudo-offspring projection
u_k: utility function
$lr = \{\eta_\mu, \eta_A, \eta_\alpha\}$: learning rate

The *t*NES algorithm requires the following inputs: fitness function, initial search distribution, a source distribution, and initial mixing coefficients. In the iteration loop, the *t*NES algorithm firstly checks for the transfer flag. If the transfer mode is not activated, λ pseudo-offspring are sampled from the target distribution and follow the original xNES procedure. Otherwise, pseudo-offspring are sampled from the mixture model (6) and their fitness are evaluated. Additionally, projection (10) is applied to all pseudo-offspring from the source distribution, after which a coordinate transformation $\boldsymbol{z}_k = A^{-1}(\widetilde{\boldsymbol{w}}_k - \mu)$ maps the projected pseudo-offspring to the natural coordinates of the target distribution (this step is required for xNES updates). Then, fitness shaping is applied. All pseudo-offspring (now $\boldsymbol{z}_k$) are sorted by their fitness, i.e.

TABLE II: THE PHYSIC-INFORMED NEURAL NETWORK AND OPTIMIZATION CONFIGURATIONS USED IN EXPERIMENTAL STUDY

		A	*B*	*C*	
Example		1D steady state convection-diffusion	2D projectile motion	Model equations of traveling waves	
				(1) Linearized Burgers (2) Nonlinear Burgers	(3) Korteweg–de Vries (KdV)
DNN architecture		$(x) - 5 - 5 - (\hat{T})$	$(t) - 3 - 3 - 3 - 3 - (\hat{x}, \hat{y})$	$(x, t) - 4 - 4 - 4 - (\hat{u})$	$(x, t) - 4 - 4 - 4 - 4 - (\hat{u})$
$\dim(\boldsymbol{w})$		45	45	56	72
no. collocation points m sampled for 1 evaluation of loss $\mathcal{L}_{PDE} + \mathcal{L}_{BC/IC}$		1000 + 2	1000 + 1	5000 + 50	10000 + 100
*t*NES & xNES setting	max. evaluation	2e5	2e5	2e5	3e5
	population size λ	20	20	20	30
	learning rate $lr = \{\eta_\delta, \eta_M, \eta_\alpha\}$	1, 5e-2, 5e-2	1, 5e-2, 5e-2	1, 1e-2, 1e-2	1, 1e-2, 1e-2
	initial search distribution $\{\mu, \sigma\mathbb{I}\}$	0, 5e-2	0, 5e-2	0, 5e-2	0, 5e-2
*t*NES transfer plan: $\{\Delta t, t_{max}\}$		2, 500	2, 500	2, 500	2, 1000
SGD (ADAM) setting	max. evaluation	2e5	2e5	2e5	3e5
	initial learning rate	5e-2	5e-2	1e-2	1e-2
	learning plan	reduce learning rate by half on plateau, with a min. learning rate set at 1e-6			

(1) For the DNN architecture, the numbers in between input and output represent the number of nodes in hidden layers. For example, $(t) - 3 - 3 - 3 - 3 - (\hat{x}, \hat{y})$ indicates a neural network with single input t, follows by 4 hidden layers with 3 nodes in each layer, and multi-output $(\hat{x}, \hat{y})$. All hidden layers, except the final hidden layer, include a bias term and use 'tanh' activation function. The final hidden layer uses 'linear' activation function and does not include a bias term.
(2) In *t*NES, $r = \sqrt{12}$ is used as the standardized radial distance for pseudo-offspring projection.
(3) For neural network optimization, in additional to the maximum number of evaluations, a target loss = 1e-9 is also set as stopping criteria.
(4) Keras package is used to perform ADAM. For ADAM parameters except those stated above, default values are used.

$f(\boldsymbol{w}_1) \geq \cdots \geq f(\boldsymbol{w}_\lambda)$, and the rank-based utility value:

$$u_k = \frac{\max\left(0, \log\left(\frac{\lambda}{2}+1\right) - \log k\right)}{\sum_{j=1}^{\lambda} \max\left(0, \log\left(\frac{\lambda}{2}+1\right) - \log j\right)}, \tag{11}$$

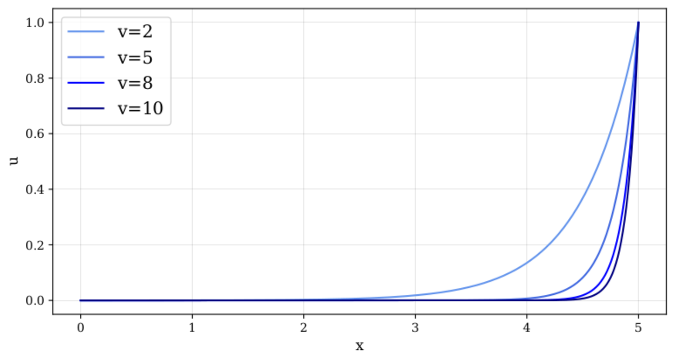
is computed to replace the k-th best fitness $f(\boldsymbol{w}_k)$. Subsequently, the algorithm follows the xNES procedure for the computation of gradient estimates and search distribution update. The *t*NES algorithm also updates the mixture model coefficients as per (9) in the transfer mode, before entering the next iteration. The optimization loop terminates once the stopping criteria, for example a target loss or maximum iterations, is met. The algorithm outputs the best found pseudo-offspring (solution) for the target problem as well as the optimized search distribution, which is added to the solved problems library as an experiential prior for future use.

There are some tunable parameters in the *t*NES algorithm; see TABLE I. The predetermined transfer plan schedules the transfer to take place after every Δt iterations, and until t_{max} iterations. Although it's safe to specify a large t_{max} since the adaptive design will auto-terminate the transfer once it's no longer beneficial, setting an appropriate t_{max} can prevent unnecessary computations. Other tunable parameters include the standardized radial distance, population size, learning rates and utility function. Like its baseline xNES algorithm, *t*NES is sensitive to these parameters, and the best settings may vary across problems.

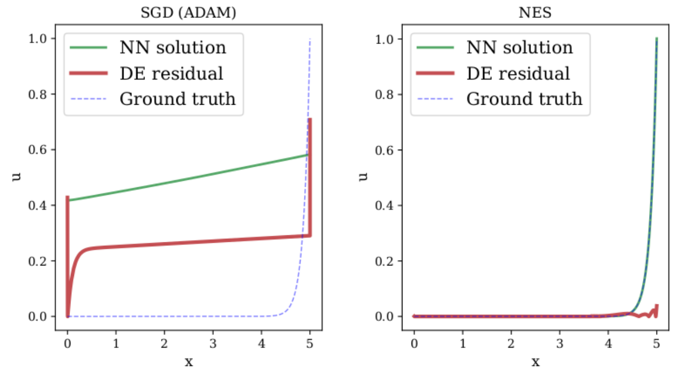
V. EXPERIMENTAL STUDY

This section contains empirical demonstrations that neuroevolution and transfer neuroevolution are noteworthy approaches for solving differential equations. Results on several pedagogical examples representative of real world phenomena: *A. 1D steady state convection-diffusion equation*, *B. 2D projectile motion equations*, and *C. model equations of traveling waves*, are presented. In each example, some parameter in the differential equation or initial condition is subject to change, forming multiple related problems to be solved. We investigate physics-informed neural networks to emulate the solution to the differential equations. The neural network architecture is fixed beforehand, without carrying out exhaustive architecture search. The selected architecture is nevertheless robust enough to produce good approximations for different problems derived from the same differential equation example. By altering the neural network weights, the differential equation as well as the prescribed initial and/or boundary conditions must be satisfied. The resultant global optimization problem can often be very challenging even if the differential equation looks simple on the surface.

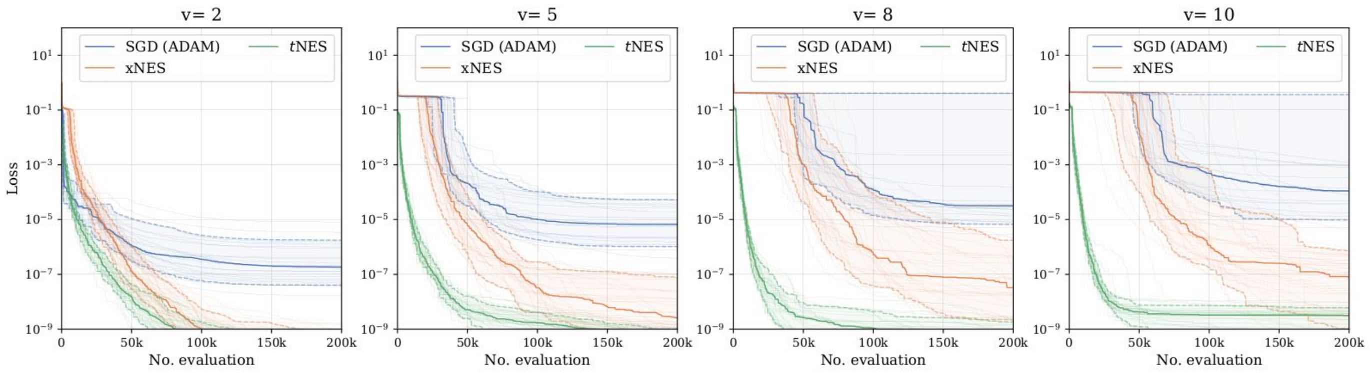
The configurations of the physics-informed neural networks and the optimization algorithms for different examples are given in TABLE II. We compare *t*NES (instantiation of transfer neuroevolution) and xNES (example of neuroevolution) with the ADAM (state-of-the-art variant of SGD) algorithm [51]. A similar setting for all 3 optimizers is used for fairness of comparisons. For each loss (fitness) evaluation, the sum of mean squared residuals from the differential equations as well as the initial and boundary conditions (2) is computed over m randomly sampled collocation points. Similar to the network architecture, we do not exhaustively search for the best optimization setting. A sufficiently robust setting that works across different problems derived from the same differential



(a) Ground truth solution for 1D convection-diffusion equation



(b) Worst SGD vs. NES solution, u=8



(c) Comparison of convergence history

Fig. 5. (a) Ground truth (analytical) solution for 1D steady state convection-diffusion equation, with v=2, 5, 8, 10 in (12). The curvature of the solution increases with velocity v and the underlying differential equation is more difficult to match, hence making it an increasingly challenging optimization problem. SGD may fail to find a good solution when u increases. For example, the SGD solution shown in (b) is not meaningful because it could not fulfill the boundary conditions, thus it induces high residual at the two boundaries of x. The worst of xNES is seen to perform notably better than SGD. In (c), the convergence trends from 30 independent SGD, xNES and *t*NES optimization runs are given. The bold lines indicate the mean convergence path, and the shaded areas indicate the interpercentile range from their 10-90th percentiles. Although the neural network loss can be optimized below 1e-9, as suggested by xNES and *t*NES results, such solutions could not be found by any of the SGD optimization runs, which always converged prematurely. The Mann-Whitney rank test concluded that both xNES and *t*NES perform better than SGD (at 200k evaluations) in all problems at 5% significance level; further, *t*NES significantly outperforms xNES when considered under a limited computational budget of 50k evaluations.

equation example is chosen. It is reckoned that the convergence of SGD (ADAM) is highly sensitive to the learning rate. Hence, a learning plan is adopted to reduce its learning rate by half on plateaus, until a minimum learning rate of 1e-6 is reached. This strategy significantly improves the robustness and performance of SGD by preventing it from being easily trapped in a plateau and also speeding up the convergence.

A. 1D Steady State Convection-Diffusion Equation

The convection-diffusion equation describes a very common phenomenon where a physical quantity, such as particles, energy, temperature, etc., are transferred inside a physical system due to two processes: convection and diffusion (also known as directional and non-directional transfer). As a motivating example, a simplified 1D steady state convection-diffusion equation is solved:

$$v \cdot u_x = k \cdot u_{xx}, \quad x \epsilon [0, L], \tag{12}$$

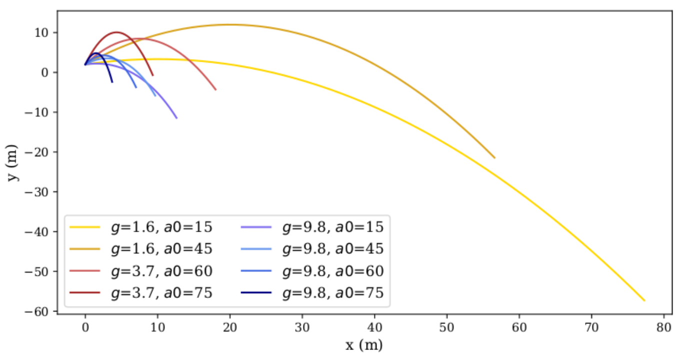
with the boundary conditions u=0, x=0; u=1, $x = L$. We are interest in the solution $u(x)$, the temperature in the spatial domain $x\epsilon[0, L]$. The velocity v, diffusion coefficient k, and domain boundary L are the problem specific constants. In this special case, the ground truth solution can be analytically derived as:

$$u(x) = \frac{1-\exp(x \cdot v/k)}{1-\exp(L \cdot v/k)}, \tag{13}$$

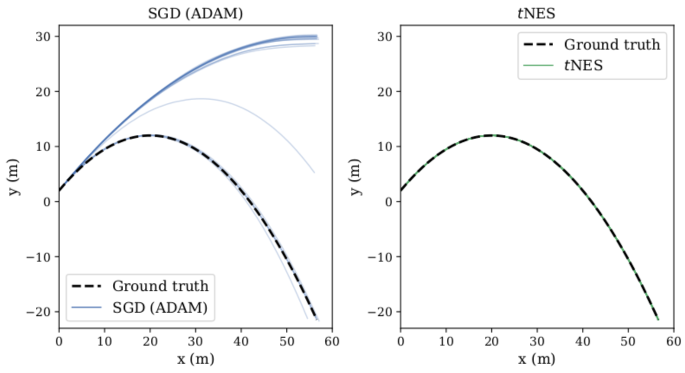
which is used to verify our optimized DNN solution. In our example, a domain boundary of L=5 and diffusion coefficient of k=1 is used. We solve the equation for different velocity values, v=2, 5, 8, 10. Classical numerical schemes may be able to solve this differential equation faster, but are notoriously prone to failure at high velocity values caused by inappropriate meshing, which gives rise to unphysical oscillations in the solution [52]. Being mesh-free, the DNN approach avoids this issue.

A physics-informed neural network with 45 tunable weights is constructed to emulate the solution u given input x. Specifically, the loss function for the 1D steady state convection-diffusion equation (12) is defined as follows:

$$\mathcal{L} = \mathcal{L}_{DE} + \mathcal{L}_{BC}$$
$$= \frac{1}{m}\sum_{i=1}^{m}(k \cdot \hat{u}_{xx} - v \cdot \hat{u}_x)^2 + [\hat{u}(0) - 0 + \hat{u}(L) - 1]. \tag{14}$$



(a) Best *t*NES optimized projectile motion solution under various scenarios



(b) Solutions from 20 independent optimization runs for 1 particular scenario

Fig. 6. (a) The projectiles under the effect of Earth (g=9.8, ρ = 1.2), Mars (g=3.7, no drag effect) and Moon (g=1.6, no drag effect) are respectively solved for t=0-2s, t=0-4.5s and t=0-10s. (b) All individual solutions given by 20 SGD and 40 *t*NES independent optimization runs for 1 particular scenario (g=1.6, a0=45, t=0-10s) are plotted and compared against the ground truth. The SGD runs lead to 3 different groups of projectiles, and deviations are observed within the 2 main groups. By knowledge transfer from a relevant prior (source distribution obtained by solving the g=3.7 at the same a0), all *t*NES runs result in an indistinguishable projectile that overlaps almost exactly with the ground truth.

Note that in this example, the loss function (14) computes the mean squared residual of the differential equation over m=1000+2 collocation points, i.e., 1000 randomly sampled points within domain $x\epsilon[0, 5]$ using Latin hypercube sampling, plus the 2 boundary points x=0, $x = L$. In addition, the 2 boundary points are matched against the boundary conditions. We refer to such computation as a single evaluation of the neural network loss. Since *t*NES, xNES and SGD algorithms are all stochastic in nature, multiple optimization runs are performed for each problem to statistically assess their results.

The results are summarized in Fig. 5. At larger u value, the solution is characterized by an extended flat region followed by steep gradient curvature near to the end of x domain (Fig. 5a). As the nonlinearity of the solution increases with velocity v, the underlying differential equation is more difficult to match. This makes the optimization problem become increasingly challenging. As a consequence, some of the solutions returned by SGD optimization are not meaningful, because they could not even fulfill the boundary conditions (Fig. 5b). These poorly optimized solutions have relatively high residual terms from the boundary conditions at two ends of x, and from the differential equation over the entire domain. In contrast, the worst solution from neuroevoluion performs much better than SGD in terms of the optimized loss (aggregated residual from the differential equation and boundary conditions) and the error against ground truth solution. For example at v=8, the aggregated residuals from the worst xNES and SGD solution are 1.3e-5 *vs.* 4.1e-1, and their mean squared errors against ground truth solution are 2.3e-7 *vs.* 2.3e-1.

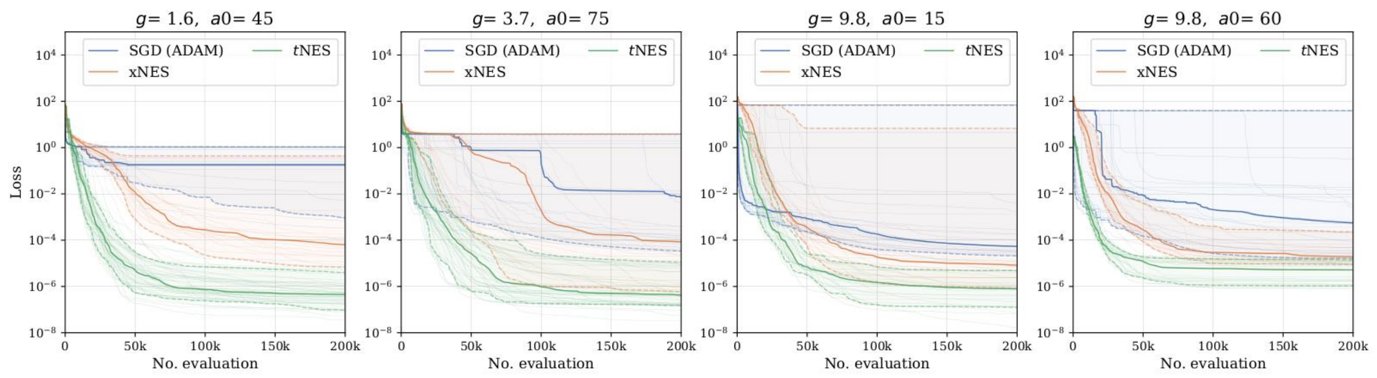
Fig. 5c compares the convergence trends of different optimizers, derived from 30 independent runs for each problem. We first compare the xNES with SGD. At lower v, SGD descends very quickly, but starts levelling off even though the loss values are far from the optimum. In contrast, xNES starts to catch up with SGD given sufficient number of evaluations, and it eventually converges (at least 2 orders of magnitude) better. As the problem gets more difficult when v increases, both xNES and SGD spend significant amount of early evaluations on a plateau before they can find a right path to descend. This is when the SGD fails to find a good solution, because on some occasions they couldn't find a right path to descend from the early plateau. Overall, xNES converges better than SGD in both speed and accuracy when u increases. This simple example demonstrates the promise of neuroevolution in solving problems where SGD gets trapped. We further demonstrate the benefit of transfer neuroevolution in Fig. 5c. For each problem, source distributions were obtained by solving the same equation with $v - 0.5$ using xNES; here, v is the velocity in the target problem. Then 30 independent *t*NES runs are performed. The proposed transfer method successfully speeds up the convergence (quickly escapes from the plateau) and also helps to achieve a better optimized solution by exploiting experiential priors. In this example, the optimized loss achieved by neuroevolution (*t*NES & xNES) are at least 2 orders of magnitude lower than SGD. What is more, under a limited computational budget of 50k evaluations, *t*NES significantly outperforms xNES (as per the Mann-Whitney rank test) across all the problems.

B. *2D Projectile Motion Equations*

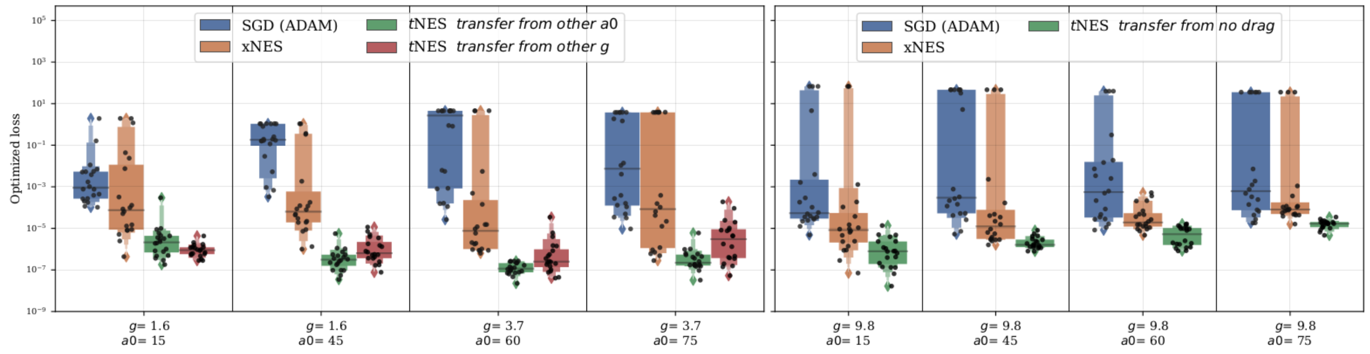
In our second example, transfer neuroevolution is applied to solve for the physics of projectile motion. Assuming a ball is thrown into the air at known launch angle a_0, initial velocity vel_0 and location (x_0, y_0), the entire projectile of the flying ball can be predicted by solving the following ordinary differential equations:

$$x_{tt} = -R \cdot x_t \quad , \quad t\epsilon[0, T], \tag{15a}$$
$$y_{tt} = -R \cdot y_t - g, \quad t\epsilon[0, T], \tag{15b}$$

subject to the initial conditions $x = x_0$, $x_t = vel_0 cos(a_0\pi/180)$, t=0, and $y = y_0$, $y_t = vel_0 sin(a_0\pi/180)$, t=0. The solution to the above differential equations gives the horizontal and vertical position $x(t)$ and $y(t)$ of the flying ball, within the time domain $t\epsilon[0, T]$. Now, let us assume the following initial conditions: (x_0, y_0)=(0, 2m), vel_0=8m/s, i.e,. the ball is released at 2m height, at an initial velocity of 8m/s (simulating a human throwing a basketball size ball with his arm). We vary



(a) Comparison of convergence history



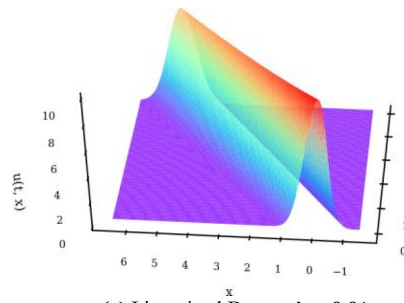
(b) Distribution of optimized loss on a test set: SGD *vs.* xNES *vs.* tNES

Fig. 7. (a) The convergence trends from 20 independent SGD and xNES optimization runs, and 40 tNES optimization runs for selected scenarios are displayed. The bold lines indicate the mean convergence path and the shaded areas indicate their 10-90th interpercentile ranges. (b) The distribution of optimized loss on a test set given by tNES, xNES and SGD for all scenarios are compared. Some scenarios (e.g., g=1.6, a0=45 & g=3.7, a0=60) are challenging to SGD and neuroevolution (xNES & tNES) could offer a much better performance. In an easy scenario (e.g., g=9.8, a0=15), all techniques converge quickly and neuroevolution descends to a lower optimized loss. For g=1.6 and g=3.7, the results (in green blocks) of transfer from the same g but other a0 (i.e., from a0=15 to a0=30, from a0=60 to a0=75, and vice versa), and the results (in red blocks) of transfer from other g (i.e., from g=3.7 to g=1.6, and from g=9.8 to g=3.7) at the same a0 are compared. For g=9.8, the source problem came from the same scenario without considering the drag effect. Comparing their optimized loss on a test set, the Mann-Whitney rank test concluded that xNES performs better than SGD in all scenarios at 5% significance level (except for g=3.7, a0=75), whereas tNES results are significantly better than both SGD and xNES.

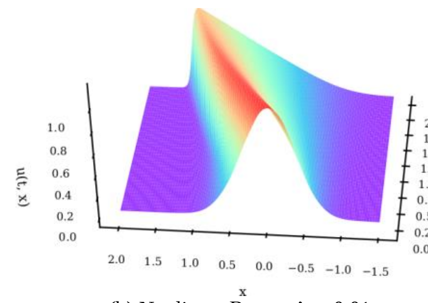
the launch angles to simulate different projectiles. In (15), R is the resistance coefficient which is related to the air density and object velocity, and g is the gravity. For simplicity, it is assumed that the ball does not spin much so the Magnus effect is neglected. Then the air resistance or drag effect is modelled by $R = CV$, where $V = \sqrt{({x_t}^2 + {y_t}^2)}$, $C = 0.5\rho C_d A/m$. ρ is the air density. C_d is the drag coefficient, $A = \pi r^2$ is the cross-sectional area of the ball, and m is its mass. The radius and mass of a basketball are typically r= 0.12 (m) and m=0.6 (kg), and the drag coefficient C_d is around 0.54 [53].

Fig. 6(a) shows the tNES optimized projectiles of the flying ball for various launch angles a_0=15, 30, 45, 75, between the time domain $t\epsilon$[0, 2s] under the effect of Earth gravity g=9.8 (m/s^2) and air ρ = 1.2 (kg/m^3). Moreover, tNES is applied to solve for the projectiles on another planet such as Mars (g=3.7, t=0-4.5s, no drag effect) or on the Moon (g=1.6, t=0-10s, no drag effect), since the same physics laws apply elsewhere in the universe. In Fig. 6(b), solutions from 20 SGD and 40 tNES optimization runs for 1 particular scenario (g=1.6, a0=45) are compared against the ground truth. The SGD produces 3 different group of projectiles, 2 groups of which are poorly optimized solutions due to being trapped in some bad local minima. There are also deviations within the groups, because the optimization runs are yet to fully converge into the minima. In contrast, the quality of tNES solutions are consistent, so they all overlap onto an indistinguishable projectile. Their median mean squared errors against the ground truth solution are 2.6e2 for SGD and 6.9e-7 for tNES. Similar pattern can be observed in other scenarios as well.

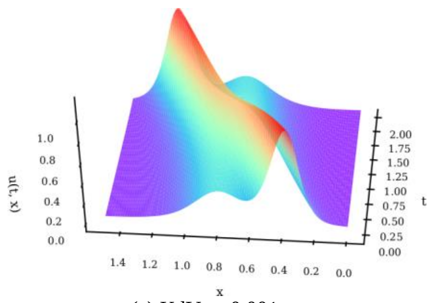
The convergence and optimized loss results given by tNES, xNES and SGD are summarized in Fig. 7. We show the convergence history for selected scenarios, and compare the optimized loss for all scenarios with a "letter value" plot [54]. Different from previous examples, for most of the problems here the xNES cannot completely avoid being trapped in a bad local minimum. Nevertheless, it offers a better chance to find better minima when compared to SGD. At the same time, tNES gives the best results by leveraging the experiential priors. These priors either come from solving a related problem scenario in the absence of drag, or a scenario with different gravitational conditions but a similar launch angle, or for one with different launch angle but similar gravity. By transferring and reusing information from such relevant sources, tNES avoids being trapped in a bad local minimum. These results demonstrate that transfer neuroevolution indeed solves



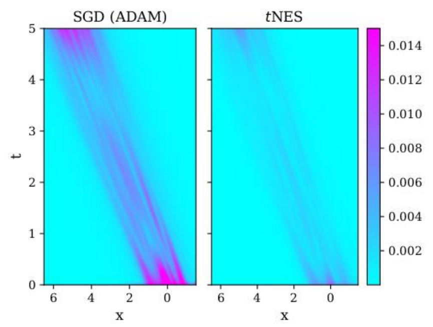
(a) Linearized Burgers' ν=0.01



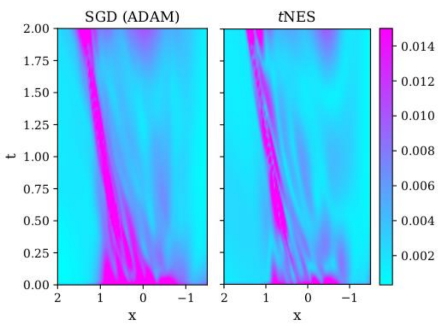
(b) Nonlinear Burgers' ν=0.01



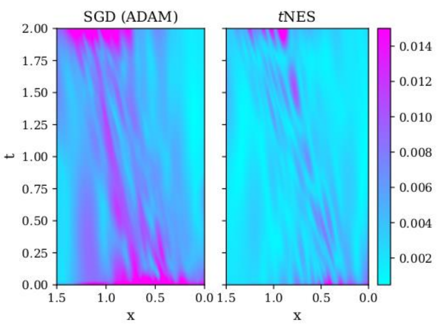
(c) KdV ν=0.001



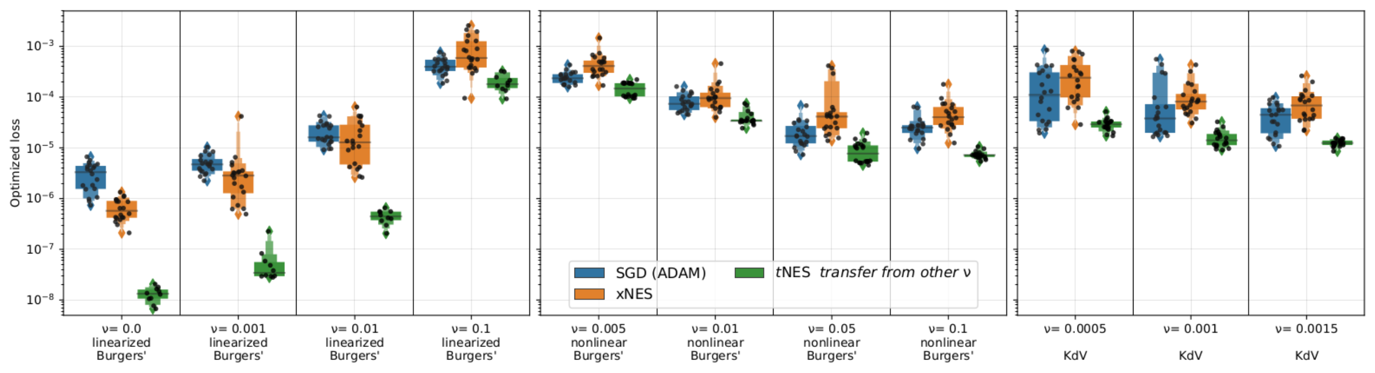
(d) Linearized Burgers' ν=0.01



(e) Nonlinear Burgers' ν=0.01



(f) KdV ν=0.001



(g) Distribution of optimized loss on a test set: SGD *vs.* xNES *vs.* *t*NES

Fig. 8. (a)-(c) Sample solution to transient PDE problems generated by *t*NES. (d)-(f) Mean absolute residual maps on a test set based on 20 SGD and *t*NES runs. The residual maps, which measures the deviation from both differential equation and initial condition, are evaluated based on 10k uniform grid points across respective computational domain. Results show that *t*NES performs better than SGD at regions near to initial state at t=0 and also regions with steep gradients for all 3 problems. (g) The distributions of optimized loss on a test set given by *t*NES, xNES and SGD for all cases are compared. Overall, SGD and xNES give a similar range of optimized loss values for the transient PDE problems. For the linearized Burgers' equation, *t*NES transfer from other ν (the same equation solved for ν=0.02) significantly improves the optimized loss. For the nonlinear Burgers' equation, *t*NES improves the optimized loss relative to the baseline xNES with the experiences transferred from other ν (for problems ν=0.005 & 0.01, source distributions obtained by solving ν=0.006 & 0.012; for problems ν=0.05 & 0.1, source distributions obtained by solving ν=0.06 & 0.12). For the KdV equation cases (for problem ν=0.0005, source distributions obtained by solving ν=0.0004 & 0.0008 ; for problem ν=0.0010, source distributions obtained by solving ν=0.0008 & 0.0016; for problem ν=0.0015, source distributions obtained by solving ν=0.0012 & 0.0016), *t*NES also improves the optimized loss relative to the baseline xNES. The tNES outperforms xNES and SGD (ADAM) at 5% significance level (as per the Mann-Whitney rank test) across all the problems considered in this example.

differential equations quicker and better.

C. *Model Equations of Traveling Waves*

In this section, transfer neuroevolution is applied to solve for several transient PDEs which describe traveling wave(s). The first model equation – Burgers' equation – can be viewed as a simplified 1D version of the Navier–Stokes equations, for understanding the dynamics of fluids and other important physics [55]. The Burgers' equation has a general form:

$$u_t + u \cdot u_x = \nu \cdot u_{xx}, \tag{16}$$

which combines the nonlinear advection and diffusion effects. The diffusion coefficient (or kinematic viscosity) ν is an important parameter for characterizing the fluid properties.

1) *Linearized Burgers' equation*

We firstly demonstrate the efficacy of transfer neuroevolution on a simpler version, linearized Burgers' equation (it can be viewed as a transient extension of our first example's 1D steady state convection-diffusion equation):

$$u_t + c \cdot u_x = \nu \cdot u_{xx}, \quad x\epsilon[-1.5, 6.5], \quad t\epsilon[0, 5], \tag{17}$$

with an initial condition $u(x,0) = 10 \cdot \exp(-(2 \cdot x)^2)$. In this setup, a single waveform is propagated at constant velocity c=1. We vary the diffusion coefficient ν=0, 0.001, 0.01, 0.1 to form different problems for transfer neuroevolution to solve; in all cases, the experiential prior is drawn from the optimized search distribution for ν=0.02. At ν=0, there is no diffusion effect, so that the initial waveform maintains its form, while at larger ν the waveform loses its magnitude over time due to the diffusion effect.

Fig. 8a presents a *t*NES solution for the case ν=0.01 (with ν =0.02 as source prior), where the propagation of the wave can be seen. In Fig. 8(d), the mean absolute residual (aggregated residual from differential equation and initial condition) for this case based on 20 *t*NES and SGD optimization runs are compared. On average, SGD gives higher residual than *t*NES near to the diffusive wave region. Higher SGD residual is also observed in capturing the initial condition. The distributions of optimized loss given by *t*NES, xNES and SGD are compared in Fig. 8(g). The optimized loss values given by SGD and xNES are in a similar range. By leveraging the experiential priors, *t*NES gives (1-2 orders of magnitude) better optimized loss than SGD and xNES.

2) *Nonlinear Burgers' equation*

Next, the full nonlinear Burgers' equation is considered:

$$u_t + u \cdot u_x = \nu \cdot u_{xx}, \ \ x\epsilon[-1.5, 2], \ \ t\epsilon[0, 2] \tag{18}$$

with an initial condition $u(x,0) = \exp(-(2 \cdot x)^2)$. Fig. 8(b) shows a *t*NES optimized solution for ν=0.01 (with ν =0.006 as source prior), illustrating the nonlinear waveform propagation with both compression and rarefaction effects. A steep gradient can be observed on one side. In Fig. 8(e), the mean absolute residual (aggregated residual from differential equation and initial condition) based on 20 independent *t*NES and SGD optimization runs for the same problem are compared. The overall results show that *t*NES performs better than SGD at the initial stages (near to t=0) and near to the steep gradient region, suggesting a more accurate solution from *t*NES. The results from multiple optimization runs for problem with ν=0.005, 0.01, 0.05, 0.1 are compared in Fig. 8(g). The optimized loss values given by both xNES and SGD are around 1e-3 to 1e-5 and could not be further reduced. This is mainly caused by the neural networks' inability to emulate the right solution at steep gradient region. Transferring from other ν, *t*NES achieves the best optimized loss among all 3 techniques.

3) *Korteweg–de Vries (KdV) equation*

The KdV equation [56] is used in physics and engineering to model the weakly nonlinear long waves (for example, waves on shallow water surfaces). It has several variants, and we consider the KdV equation in the form:

$$u_t + u \cdot u_x = \nu \cdot u_{xxx}, \ \ x\epsilon[0, 1.5], \ \ t\epsilon[0, 2] \tag{19}$$

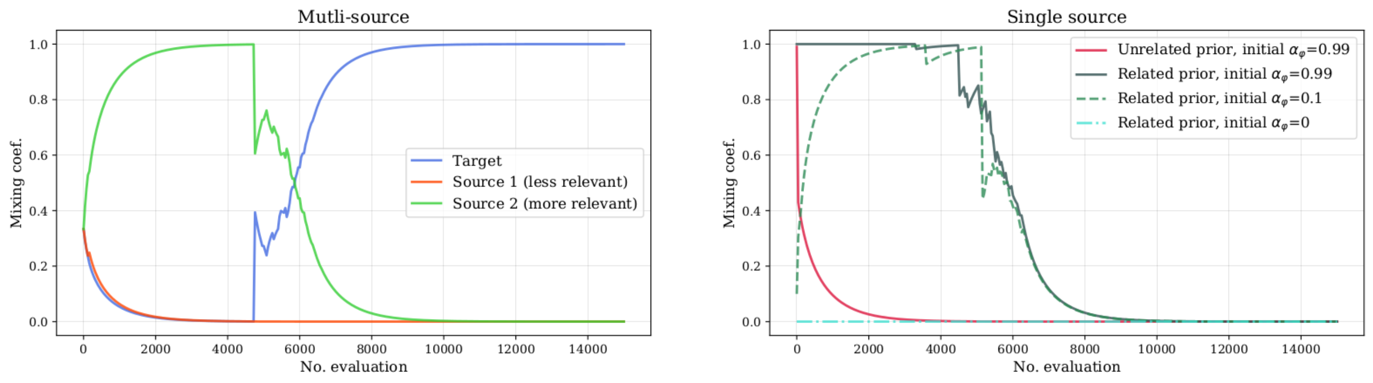
which consists of a dispersive coefficient ν. The solution of (19) describes the height of the wave at position x and time t. We consider the following initial condition [57]: $(x,0) = 3\mathrm{c}_1\mathrm{sech}^2\mathrm{a}_1(x - x_1) + 3\mathrm{c}_2\mathrm{sech}^2\mathrm{a}_2(x - x_2)$. Specifically, c_1=0.3, c_2=0.1, x_1=0.4, x_2=0.8, a_1=$0.5\sqrt{\mathrm{c}_1/\nu}$, and a_2=$0.5\sqrt{\mathrm{c}_2/\nu}$. With this initial condition, the equation simulates the collision of 2 waves of different magnitudes traveling from different locations. These 2 waves gradually lose their magnitudes while they approach each other, and then collide. After the collision, they split and regain their magnitudes. Fig. 8(c) shows such interaction for ν =0.001 using the solution optimized by *t*NES (with ν =0.0008 as source prior). Fig. 8(f) compares the mean absolute residual (aggregated residual from differential equation and initial condition) between *t*NES and SGD solutions for the same case, based on 20 independent optimization runs. The results suggest a more accurate solution from *t*NES across the computational domain. Fig 8(g) shows the distribution of the optimized loss based on multiple *t*NES, xNES and SGD optimization runs for dispersive coefficient ν=0.0005, 0.0010, 0.0015. The optimized loss values given by xNES are around 1e-3 to 5e-5, while SGD doesn't offer significantly better results. Transferring from other ν, *t*NES improves the optimized loss significantly in comparison to the baseline xNES.

D. *Study of Mixing Coefficients in tNES*

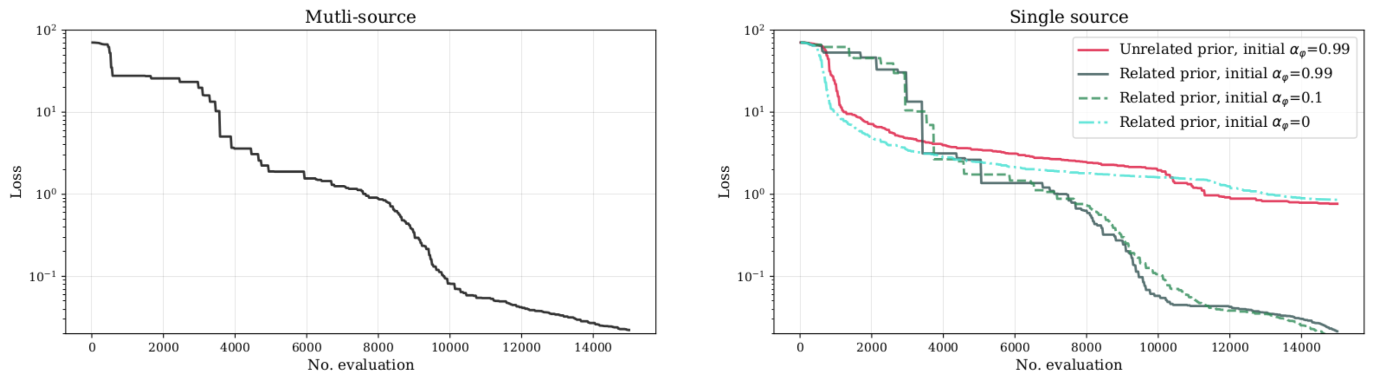
In this section, we provide visualization and discussion on how the mixing coefficients change during the evolutionary process, under the proposed mixture model-based adaptive transfer. Taking the 2D projectile motion as example, the target problem is to solve the projectile motion under the effect of moon gravity g=1.6, with an initial launch angle a0=45.

Firstly, a multi-source setup is considered, where one of source is less relevant (optimized search distribution for g=9.8, a0=45) to the target than the other (optimized search distribution for g=3.7, a0=45). By default, the mixture model for transfer neuroevolution is initialized with equal mixing coefficients for the target and all source distribution components. As shown in the left panel of Fig. 9a, the mixing coefficient of the beneficial source quickly ascends to 1 at the early evolutionary stage, while suppressing the mixing coefficients from the less relevant source. This shows that the mixture model-based transfer is capable of jointly processing multiple sources and selecting the one that is most relevant to the target. After a certain point however, the source no longer remains competitive with the evolved target search distribution, which causes its mixing coefficient to gradually drop. Eventually, the target search distribution takes over the mixture model and all the source components are deactivated. Due to the directional bias induced by the source priors, the search progresses at a faster rate towards a better solution.

Next, the sensitivity of initial mixing coefficient αs' is investigated under the single source setup. We consider the scenario where the single source from a related prior (optimized search distribution for g=3.7, a0=45), and is initialized at different levels of α_φ= 0.99, 0.1, 0. As shown in the right panel of Fig. 9a, α_φ quickly ascends to dominate the mixture model at the early evolutionary stage, even when it has a small initial value. That is, the proposed mixture model-based adaptive



(a) History of mixing coefficient αs' in *t*NES



(b) Convergence history

Fig. 9. (a) In the multi-source plot, the history of mixing coefficient αs' from all mixture model components during a *t*NES search process are shown. In the single source plot, the history of mixing coefficient of a single source is shown under different initial values of α_φ. Their respective convergence trends between 0 and 15,000 evaluations are shown in (b), which highlights that effective transfer neuroevolution leads to better convergence.

transfer method is not very sensitive to the initial αs' setting. The only exception is when the initial α_φ is set to 0, since in this case pseudo-offspring will never be drawn from the source distribution. Thus, transfer never occurs. On the other hand, if the single source forms an unrelated prior (far from the optimum target distribution), its mixing coefficient quickly descends to 0 so that negative transfer is avoided. As a result, the search has a similar convergence trend as the no-transfer scenario (right panel of Fig. 9b).

VI. Conclusions

In conclusion, this paper demonstrated neuroevolution as a notable approach for solving differential equations, where the problem has been transformed to one of global optimization of physics-informed neural networks. In such problems, we merit the accuracy of the solution, which shall be produced by a better optimized neural network. Gradient descent methods such as SGD may not always be the best approach, because they are prone to premature convergence. Therefore, we introduce neuroevolution, which naturally adapts a parallel exploration of diverse solutions during the optimization search to overcome local optima and to achieve better optimized neural networks. We empirically demonstrated that an implementation of the state-of-the-art NES algorithm could outperform SGD in many differential equations to give more accurate solutions.

Moreover, a novel transfer neuroevolution method which is suitable for general probabilistic distribution-based evolution strategies is proposed. It features a mixture model-based adaptive transfer mechanism to systematically transfer useful knowledge from relevant source problems, while protecting the search against negative transfer. In practice, it is very common for many similar differential equation problems to occur under different environments and boundary conditions; as such, we expect transfer neuroevolution to shine in this domain. We empirically demonstrated that transfer neuroevolution may not only improve the convergence speed but also improve the solution accuracy in comparison to the baseline neuroevolution algorithm.

In our experimental study, neuroevolution nevertheless starts losing its competitive advantage to SGD (ADAM) for more complex differential equations. On this account, it is important for future research to further improve the effectiveness of neuroevolution in terms of: (1) handling much larger neural networks, such that the solution of complex differential equations can be accurately emulated; and (2) simultaneously searching for the best network architecture in addition to the neural network weights [58, 59]. Another interesting idea is the development of multi-objective, multi-task neuroevolution [60, 61] in the context of physics-informed neural networks. We expect these approaches to have a good synergy with the recently proposed domain decomposition strategy [62, 63], where separate neural networks are used to approximate different sub-domains of a differential equation.

References

[1] W. Ames, *Numerical methods for partial differential equations*. Boston: Academic Press, 1992.

[2] K. W. Morton and D. F. Mayers, *Numerical Solution of Partial Differential Equations*. Cambridge University Press, apr 2005.

[3] M. Heath, *Scientific computing : an introductory survey*. Philadelphia, Pennsylvania: Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104, 2018.

[4] R. Mattheij, *Partial differential equations : modeling, analysis, computation*. Philadelphia: Society for Industrial and Applied Mathematics, 2005.

[5] L. Debnath, *Nonlinear partial differential equations for scientists and engineers*. Boston: Birkhaı̀ˆuser, 2012.

[6] H. Wang, *Modeling information diffusion in online social networks with partial differential equations*. Cham: Springer, 2020.

[7] J. Ankudinova and M. Ehrhardt, "On the numerical solution of nonlinear blackscholes equations," *Computers & Mathematics with Applications*, vol. 56, no. 3, pp. 799–812, aug 2008.

[8] H. Lee and I. S. Kang, "Neural algorithm for solving differential equations," *Journal of Computational Physics*, vol. 91, no. 1, pp. 110–131, nov 1990.

[9] A. Meade and A. Fernandez, "Solution of nonlinear ordinary differential equations by feedforward neural networks," *Mathematical and Computer Modelling*, vol. 20, no. 9, pp. 19–44, nov 1994.

[10] ——, "The numerical solution of linear ordinary differential equations by feedforward neural networks," *Mathematical and Computer Modelling*, vol. 19, no. 12, pp. 1–25, jun 1994.

[11] M. W. M. G. Dissanayake and N. Phan-Thien, "Neural-network-based approximations for solving partial differential equations," *Communications in Numerical Methods in Engineering*, vol. 10, no. 3, pp. 195–201, mar 1994.

[12] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural network methods in quantum mechanics," *Computer Physics Communications*, vol. 104, no. 1-3, pp. 1–14, aug 1997.

[13] ——, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[14] I. Lagaris, A. Likas, and D. Papageorgiou, "Neural-network methods for boundary value problems with irregular boundaries," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041–1049, 2000.

[15] K. McFall and J. Mahan, "Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions," *IEEE Transactions on Neural Networks*, vol. 20, no. 8, pp. 1221–1233, aug 2009.

[16] K. Rudd, G. D. Muro, and S. Ferrari, "A constrained backpropagation approach for the adaptive solution of partial differential equations," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 3, pp. 571–584, mar 2014.

[17] K. Rudd and S. Ferrari, "A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks," *Neurocomputing*, vol. 155, pp. 277–285, may 2015.

[18] J. Berg and K. Nyström, "A unified deep artificial neural network approach to partial differential equations in complex geometries," *Neurocomputing*, vol. 317, pp. 28–41, nov 2018.

[19] J. Sirignano and K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1339–1364, dec 2018.

[20] C. Anitescu, E. Atroshchenko, N. Alajlan, and T. Rabczuk, "Artificial neural network methods for the solution of second order boundary value problems," *Computers, Materials & Continua*, vol. 59, no. 1, pp. 345–359, 2019.

[21] N. Geneva and N. Zabaras, "Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks," *Journal of Computational Physics*, vol. 403, p. 109056, feb 2020.

[22] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, "Adaptive activation functions accelerate convergence in deep and physics-informed neural networks," *Journal of Computational Physics*, vol. 404, p. 109136, mar 2020.

[23] Z. Mao, A. D. Jagtap, and G. E. Karniadakis, "Physics-informed neural networks for high-speed flows," *Computer Methods in Applied Mechanics and Engineering*, vol. 360, p. 112789, mar 2020.

[24] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, feb 2019.

[25] J. Peiró and S. Sherwin, "Finite difference, finite element and finite volume methods for partial differential equations," in *Handbook of Materials Modeling*. Springer Netherlands, 2005, pp. 2415–2446.

[26] Y. Chen, L. Lu, G. E. Karniadakis, and L. D. Negro, "Physics-informed neural networks for inverse problems in nano-optics and metamaterials," *Optics Express*, vol. 28, no. 8, p. 11618, apr 2020.

[27] M. Raissi, "Deep hidden physics models: Deep learning of nonlinear partial differential equations," *Journal of Machine Learning Research*, vol. 19, no. 25, pp. 1–24, 2018. [Online]. Available: http://jmlr.org/papers/v19/18-046.html

[28] Y. Yang and P. Perdikaris, "Adversarial uncertainty quantification in physics-informed neural networks," *Journal of Computational Physics*, vol. 394, pp. 136–152, oct 2019.

[29] M. Raissi, Z. Wang, M. S. Triantafyllou, and G. E. Karniadakis, "Deep learning of vortex-induced vibrations," *Journal of Fluid Mechanics*, vol. 861, pp. 119–137, dec 2018.

[30] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, vol. 367, no. 6481, pp. 1026–1030, jan 2020.

[31] G. Kissas, Y. Yang, E. Hwuang, W. R. Witschey, J. A. Detre, and P. Perdikaris, "Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow MRI data using physics-informed neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 358, p. 112623, jan 2020.

[32] Y.-S. Ong and A. Gupta, "AIR5: Five pillars of artificial intelligence research," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 3, no. 5, pp. 411–415, oct 2019.

[33] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, jan 2019.

[34] J.-B. Mouret and S. Doncieux, "Encouraging behavioral diversity in evolutionary robotics: An empirical study," *Evolutionary Computation*, vol. 20, no. 1, pp. 91–133, mar 2012.

[35] X. Zhang, J. Clune, and K. O. Stanley, "On the relationship between the openai evolution strategy and stochastic gradient descent."

[36] J. Lehman, J. Chen, J. Clune, and K. O. Stanley, "Es is more than just a traditional finite-difference approximator."

[37] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *Journal of Machine Learning Research*, vol. 15, no. 27, pp. 949–980, 2014. [Online]. Available: http://-jmlr.org/papers/v15/wierstra14a.html

[38] A. Gupta, Y.-S. Ong, and L. Feng, "Insights on transfer optimization: Because experience is the best teacher," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 51–64, feb 2018.

[39] B. Da, A. Gupta, and Y.-S. Ong, "Curbing negative influences online for seamless transfer evolutionary optimization," *IEEE Transactions on Cybernetics*, vol. 49, no. 12, pp. 4365–4378, dec 2019.

[40] B. Da, "Methods in multi-source data-driven transfer optimization," Ph.D. dissertation, Nanyang Technological University, 2019. [Online]. Available: https://hdl.handle.net/10356/136964

[41] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, jun 2001.

[42] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning."

[43] W. E and B. Yu, “The deep ritz method: A deep learning-based numerical algorithm for solving variational problems,” *Communications in Mathematics and Statistics*, vol. 6, no. 1, pp. 1–12, feb 2018.

[44] S. Goswami, C. Anitescu, S. Chakraborty, and T. Rabczuk, “Transfer learning enhanced physics informed neural network for phase-field modeling of fracture,” *Theoretical and Applied Fracture Mechanics*, vol. 106, p. 102447, apr 2020.

[45] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, “Automatic differentiation in machine learning: a survey,” *Journal of Machine Learning Research*, vol. 18, no. 153, pp. 1–43, 2018. [Online]. Available: http://jmlr.org/papers/v18/17-468.html

[46] C. Igel, “Neuroevolution for reinforcement learning using evolution strategies,” in *The 2003 Congress on Evolutionary Computation, 2003. CEC 03.* IEEE, 2003.

[47] P. Pagliuca and S. Nolfi, “Robust optimization through neuroevolution,” *PLOS ONE*, vol. 14, no. 3, p. e0213193, mar 2019.

[48] C. Colas, V. Madhavan, J. Huizinga, and J. Clune, “Scaling MAP-elites to deep neuroevolution,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference - GECCO 20*. ACM Press, 2020.

[49] T. Schaul, T. Glasmachers, and J. Schmidhuber, “High dimensions and heavy tails for natural evolution strategies,” in *Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO 11*. ACM Press, 2011.

[50] T. Glasmachers, T. Schaul, S. Yi, D. Wierstra, and J. Schmidhuber, “Exponential natural evolution strategies,” in *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO 10*. ACM Press, 2010.

[51] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[52] A. Gupta, “Numerical modelling and optimization of non-isothermal, rigid tool liquid composite moulding processes,” Ph.D. dissertation, The University of Auckland, 2013.

[53] H. Okubo and M. Hubbard, “Identification of basketball parameters for a simulation model,” *Procedia Engineering*, vol. 2, no. 2, pp. 3281–3286, jun 2010.

[54] H. Hofmann, H. Wickham, and K. Kafadar, “Letter-value plots: Boxplots for large data,” *Journal of Computational and Graphical Statistics*, vol. 26, no. 3, pp. 469–477, jul 2017.

[55] U. Frisch and J. Bec, “Burgulence.”

[56] D. J. Korteweg and G. de Vries, “XLI. on the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 39, no. 240, pp. 422–443, may 1895.

[57] Y. Yuan, J. Li, L. Li, F. Jiang, X. Tang, F. Zhang, S. Liu, J. Goncalves, H. U. Voss, X. Li, J. Kurths, and H. Ding, “Machine discovery of partial differential equations from spatiotemporal data.”

[58] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, jun 2002.

[59] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, “Pathnet: Evolution channels gradient descent in super neural networks.”

[60] O. Abramovich and A. Moshaiov, “Multi-objective topology and weight evolution of neuro-controllers,” in *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, jul 2016.

[61] S. Künzel and S. Meyer-Nieberg, “Evolving artificial neural networks for multi-objective tasks,” in *Applications of Evolutionary Computation*. Springer International Publishing, 2018, pp. 671–686.

[62] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, “Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems,” *Computer Methods in Applied Mechanics and Engineering*, vol. 365, p. 113028, jun 2020.

[63] K. Li, K. Tang, T. Wu, and Q. Liao, “D3m: A deep domain decomposition method for partial differential equations,” *IEEE Access*, vol. 8, pp. 5283–5294, 2020.