Transfer Learning with Physics-Informed Neural Networks for Efficient Simulation of Branched Flows

Raphaël Pellegrin
Harvard University
raphaelpellegrin@fas.harvard.edu

Blake Bullwinkel
Harvard University
jbullwinkel@fas.harvard.edu

Marios Mattheakis
Harvard University
mariosmat@seas.harvard.edu

Pavlos Protopapas
Harvard University
pavlos@seas.harvard.edu

Abstract

Physics-Informed Neural Networks (PINNs) offer a promising approach to solving differential equations and, more generally, to applying deep learning to problems in the physical sciences. We adopt a recently developed transfer learning approach for PINNs and introduce a multi-head model to efficiently obtain accurate solutions to nonlinear systems of differential equations. In particular, we apply the method to simulate stochastic branched flows, a universal phenomenon in random wave dynamics. We compare the results achieved by feed forward and GAN-based PINNs on two physically relevant transfer learning tasks and show that our methods provide significant computational speedups in comparison to standard PINNs trained from scratch.

1 Introduction

Differential equations are used to describe a plethora of phenomena in the physical sciences but most cannot be solved analytically. Traditionally, numerical methods have been used to approximate solutions to differential equations. Recently, Physics-Informed Neural Networks (PINNs) have emerged as an attractive alternative offering several compelling advantages. In particular, PINNs: provide solutions that are in closed form, offer a more accurate interpolation scheme [7], are more robust to the "curse of dimensionality" [4, 5, 13, 14], and do not accumulate numerical errors [6, 11].

PINNs are typically trained to solve only a single configuration of a given system (e.g., a single initial condition or set of system parameters) at once, making their practical use computationally inefficient. More recently, it was shown that one-shot transfer learning can be used to obtain accurate solutions to linear systems of ordinary differential equations (ODEs) and partial differential equations (PDEs), thereby eliminating the need to train the network from scratch for a new linear system [3, 9].

In this work, we build upon [3] by proposing a method that can be applied to non-linear systems. This method consists of two phases. First, we train a base neural network with multiple output heads, solving the system for a range of different configurations (e.g., initial conditions or potentials). We thus learn a representative basis that captures the underlying dynamics. Second, we freeze the weights of the base network and fine-tune new linear heads on a secondary transfer learning task. In doing so, we adapt the pre-trained base from one task to another and cut computational costs significantly. We demonstrate the efficacy of our approach using a system of non-linear ODEs that

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

describes the trajectory of a particle through a weak random potential and can be used to model a universal phenomenon called branched flow [2, 12].[1]

2 Background

2.1 PINN Models

This work uses PINNs that are trained in an unsupervised manner, as detailed below. We compare the performance achieved by feed forward neural networks (FFNN) and GAN models.

FFNN: The standard unsupervised neural network approach was introduced by Lagaris et al. [7] and can be used to solve differential equations of the form

$$\mathcal{L}\left(u(t, \mathbf{x})\right) = 0, \tag{1}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$, $u : \mathbb{R}^{n+1} \to \mathbb{R}$ and $\mathcal{L}$ is a differential operator. During training, we sample $(t, \mathbf{x})$ from the domain of the equation D and use this vector as input to a FFNN, which outputs the neural solution $u_\theta(t, \mathbf{x})$. We re-parametrise this output into $\tilde{u}_\theta(t, \mathbf{x})$ to satisfy initial and boundary conditions exactly. Using automatic differentiation, we can compute the derivatives of this output with respect to each of the independent variables and build the loss by summing the squared residual over M training points,

$$\frac{1}{M} \sum_{(t,\mathbf{x}) \in D} \mathcal{L}\left(\tilde{u}_\theta(t, \mathbf{x})\right)^2. \tag{2}$$

Note that if the network $\tilde{u}_\theta$ perfectly satisfies Equation 1, then Equation 2 will be zero.

DEQGAN: Bullwinkel et al. [1] noted that there is no theoretical reason to use the L_2 norm of the residuals over any other loss function and proposed DEQGAN, which extends the FFNN method to GANs and can be thought of as "learning the loss function." Rather than computing a loss over the equation residuals, DEQGAN labels these vectors "fake" data samples and zero-centered Gaussian noise as "real" data samples. As the discriminator gets better at classifying these samples, the generator $\tilde{u}_\theta$ is forced to propose solutions such that the equation residuals are increasingly indistinguishable from a vector of zeros, thereby approximating the solution to the differential equation.

2.2 Transfer Learning with Multi-Head PINNs

Figure 1 illustrates the multi-head architecture that we apply to FFNN and DEQGAN models to perform transfer learning. The output of the base neural network is passed to heads $h_1, ..., h_L$, each of which corresponds to the solution to the system at a particular initial condition. Importantly, this architecture can be used to apply transfer learning to non-linear problems; in this work, we consider one such system of ODEs that is used to model particles moving through a weak random potential.

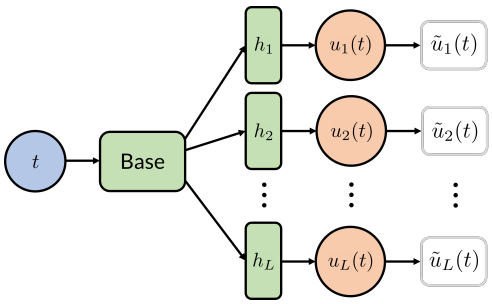


Figure 1: Multi-head PINN architecture. Each output head h_l is responsible for generating the solution to the l^{th} initial condition.

Transfer learning with multi-head PINNs is performed in two stages. First, we train the multi-head model on a given set of initial conditions until convergence. Next, we freeze the weights of the base network and fine-tune only the output heads on a second set of initial conditions. As detailed below, we use this procedure to perform two transfer learning tasks: 1) Initial Condition Transfer, which involves fine-tuning the heads on initial conditions that were not used to train the base. 2) Potential Transfer Learning, an even more challenging task that allows us to obtain solutions for new initial conditions and a different potential than the one used to train the base. Our results on these tasks suggest that the base is able to learn highly general properties of the system.

[1] All code is publicly available at https://github.com/RaphaelPellegrin/Transfer-Learning-with-PINNs-for-Efficient-Simulation-of-Branched-Flows.git.

3 Experimental Results

3.1 Branched Flow

Stochastic branched flow is a universal wave phenomenon that occurs when waves propagate in random environments. Branching has been observed in tsunami waves [2], electronic flows in graphene [12], and electromagnetic waves in gravitational fields [8]. We can model a two dimensional branched flow by considering a particle with position $\mathbf{x} = (x, y)$ and velocity $\mathbf{p} = (p_x, p_y)$, both functions of time t, traveling through a weak random potential $V(x, y)$. With the Hamiltonian $H(\mathbf{x}, \mathbf{p}) = ||\mathbf{p}||_2^2/2 + V(\mathbf{x})$ we obtain Hamilton's equations, given by the following system of ODEs

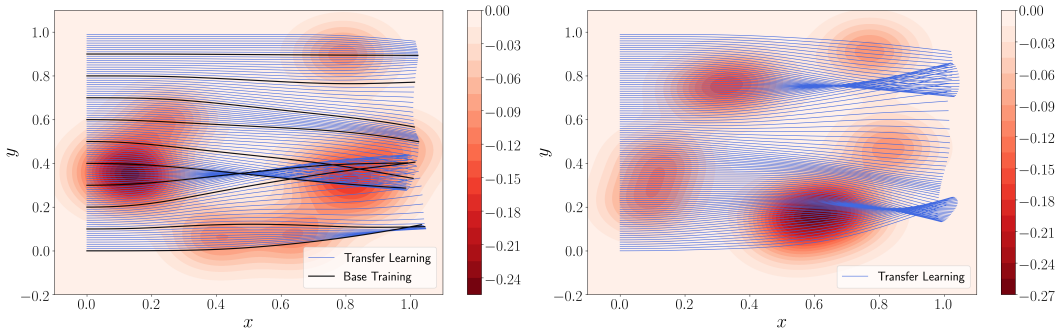
$$\begin{cases} \dot{x}(t) = p_x(t) \\ \dot{y}(t) = p_y(t) \\ \dot{p_x}(t) = -\dfrac{\partial V(x(t), y(t))}{\partial x} \\ \dot{p_y}(t) = -\dfrac{\partial V(x(t), y(t))}{\partial y}, \end{cases} \tag{3}$$

For a plane wave, the initial conditions at $t = 0$ can be chosen as $(x(0), y(0), p_x(0), p_y(0)) = (0, y(0), 1, 0)$ [12].

We build random potentials by summing K randomly distributed Gaussian functions with covariance matrix $\sigma^2 I_2 \in \mathbb{R}^{2\times 2}$ and means $\mu_i \in \mathbb{R}^2$, $i = 1, \ldots, K$, and scaling the result by $-A$, where $A \in \mathbb{R}^+$, as in [12]. That is,

$$V(\mathbf{x}) = -\frac{A}{2\pi\sigma^2} \sum_{i=1}^{K} \exp\left(-\frac{1}{2\pi\sigma^2} ||\mathbf{x} - \mu_i||_2^2\right). \tag{4}$$

In the experiments presented below, we use $K = 10,\ A = 0.1,\ \sigma = 0.1$.



(a) Initial Condition Transfer

(b) Potential Transfer Learning

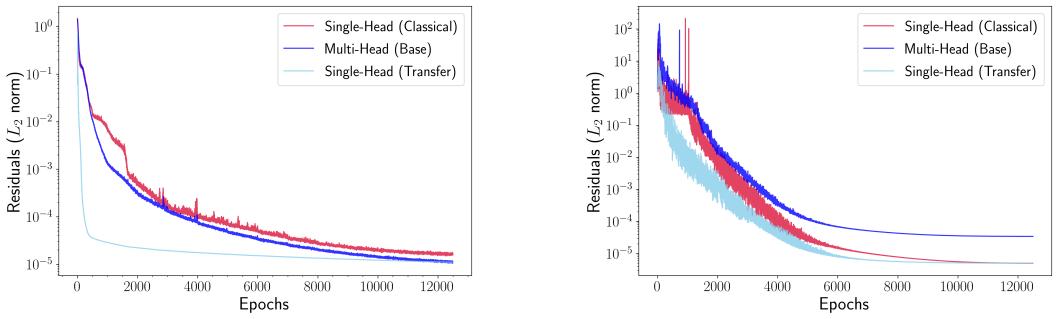
Figure 2: Particle trajectories through weak potentials generated by FFNN models. Black lines correspond to the 11 initial conditions used for base training, while blue lines show the solutions obtained for 100 evenly spaced initial conditions via transfer learning after freezing the base. The color bars indicate the value of the potential.

3.2 Details for Hamilton's Equations

For both the FFNN and DEQGAN models, we use networks with a base consisting of 5 hidden layers and 40 nodes. We then use L linear layers for the heads. Each head is responsible for the solution to one ray, i.e., one initial condition $\mathbf{z}_l(0) = (0, y_l(0), 1, 0)$, where $l = 1, \ldots, L$, and has four outputs corresponding to x, y, p_x and p_y. For head l, we denote the outputs as $u_{l,1}$, $u_{l,2}$, $u_{l,3}$ and $u_{l,4}$. We use the initial value re-parameterization proposed by Mattheakis et al. [10]

$$\tilde{u}_{l,i}(t) = [\mathbf{z}_l(0)]_i + \left(1 - e^{-t}\right) u_{l,i}(t), \quad l = 1, \ldots, L; i = 1, \ldots, 4 \tag{5}$$

which forces the proposed solution to be exactly $\mathbf{z}_l(0)$ when $t = 0$ and decays this constraint exponentially in t.



(a) FFNN (b) DEQGAN

Figure 3: Epochs vs. L_2 norm of the equation residuals for single-head (classical), multi-head (base training) and single-head (Initial Condition Transfer) runs using FFNN and DEQGAN models. The multi-head model losses are computed by averaging over the 11 heads.

Table 1: Comparison between the computational efficiency (measured in epochs/sec) of training classical single-head PINNs and performing Initial Condition Transfer for FFNN and DEQGAN.

	Epochs per second		
	Single-Head (Classical)	Multi-Head (Base)	Single-Head (Transfer)
FFNN	35.27	4.13	**42.34**
DEQGAN	8.66	1.77	**17.09**

3.3 Transfer Learning Results

Our first transfer learning task, Initial Condition Transfer, allows us to efficiently obtain solutions to the system for many initial conditions. We used multi-head models to train the base networks on $L = 11$ initial conditions $y_l(0) = 0.0, 0.1, \ldots, 1.0$ and performed single-head transfer learning on 100 evenly-spaced initial conditions in $[0, 1]$ while keeping the potential fixed. All experiments were performed on a Microsoft Surface laptop with Intel i7 CPU. Figure 2a shows the ray trajectory solutions corresponding to the initial conditions used for base training (black) and transfer learning (blue) obtained with the FFNN model. These trajectories also illustrate branched flows.

In Figure 3, we compare the losses achieved by the FFNN and DEQGAN models during base training and transfer learning. We also show the residuals for classical models that do not leverage transfer learning. Notably, we see that single-head models that use transfer learning converge more rapidly than those trained from scratch. Further, Table 1 shows that each epoch of transfer learning (bold) is also significantly faster. This is to be expected because transfer learning involves only fine-tuning linear heads, rather than training an entire base network.

Our second transfer learning task, Potential Transfer Learning, utilizes the same pre-trained base described above. This task, however, changes not only the initial conditions, but also the potential (Equation 4). More specifically, we constructed a new potential by randomly sampling ten new Gaussian means. To avoid significantly altering the statistical properties of the system, we used the same values of σ and A. Figure 2b visualizes the ray trajectories obtained using this method and suggests that the multi-head models are, indeed, able to learn highly general bases for the system.

4 Conclusion

In this paper, we propose a multi-head PINN architecture and a framework for performing transfer learning with non-linear systems of differential equations. In particular, we simulate branched flows with Hamilton's equations and demonstrate that our method significantly reduces the computational cost of obtaining solutions to many initial conditions in comparison to FFNN and GAN-based models trained from scratch, without sacrificing accuracy. Finally, we show that base networks trained using our method can transfer to new initial conditions and new potentials at the same time, indicating that our method is able to learn highly general statistical properties of the system.

5 Broader Impact

This paper presents techniques that we hope will increase the utility of PINNs in real-world applications. In particular, the transfer learning procedure explored in this work trains a base neural network on different configurations of the system of equations, thereby forcing the network to learn general properties of the solutions and providing possible insights into the underlying physical problem. Beyond computational speedups, we hope that this contributes to broader efforts within the research community to make PINNs more interpretable, and ultimately more widely adopted. We believe that future work focused on the theoretical foundations of PINNs will help cement these models as a third pillar within the study of differential equations, alongside analytical and numerical methods.

References

[1] Bullwinkel, B., Randle, D., Protopapas, P., & Sondak, D. (2022). Deqgan: Learning the loss function for pinns with generative adversarial networks. arXiv preprint arXiv:2209.07081.

[2] Degueldre, H., Metzger, J. J., Geisel, T., & Fleischmann, R. (2016). Random focusing of tsunami waves. Nature Physics, 12(3), 259–262.

[3] Desai, S., Mattheakis, M., Joy, H., Protopapas, P., & Roberts, S. (2021). One-shot transfer learning of physics-informed neural networks. arXiv preprint arXiv:2110.11286.

[4] Grohs, P., Hornung, F., Jentzen, A., & Von Wurstemberger, P. (2018). A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. arXiv preprint arXiv:1809.02362.

[5] Han, J., Jentzen, A., & E, W. (2018). Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences, 115(34), 8505–8510.

[6] Jin, H., Mattheakis, M., & Protopapas, P. (2020). Unsupervised neural networks for quantum eigenvalue problems. arXiv preprint arXiv:2010.05075.

[7] Lagaris, I. E., Likas, A., & Papageorgiou, D. G. (1998). Neural network methods for boundary value problems defined in arbitrarily shaped domains. arXiv preprint cs/9812003.

[8] Loutsenko, I. (2018). On the role of caustics in solar gravitational lens imaging. Progress of Theoretical and Experimental Physics, 2018(12), 123A02.

[9] Mattheakis, M., Joy, H., & Protopapas, P. (2021). Unsupervised reservoir computing for solving ordinary differential equations. arXiv preprint arXiv:2108.11417.

[10] Mattheakis, M., Protopapas, P., Sondak, D., Di Giovanni, M., & Kaxiras, E. (2019). Physical symmetries embedded in neural networks. arXiv preprint arXiv:1904.08991.

[11] Mattheakis, M., Sondak, D., Dogra, A. S., & Protopapas, P. (2022). Hamiltonian neural networks for solving equations of motion. Phys. Rev. E, 105, 065305.

[12] Mattheakis, M., Tsironis, G., & Kaxiras, E. (2018). Emergence and dynamical properties of stochastic branching in the electronic flows of disordered dirac solids. EPL (Europhysics Letters), 122(2), 27003.

[13] Raissi, M. (2018). Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. arXiv preprint arXiv:1804.07010.

[14] Sirignano, J. & Spiliopoulos, K. (2018). Dgm: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics, 375, 1339–1364.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]