TT-PINN: A Tensor-Compressed Neural PDE Solver for Edge Computing

Ziyue Liu [*1] **Xinling Yu** [*2] **Zheng Zhang** [2]

Abstract

Physics-informed neural networks (PINNs) have been increasingly employed due to their capability of modeling complex physics systems. To achieve better expressiveness, increasingly larger network sizes are required in many problems. This has caused challenges when we need to train PINNs on edge devices with limited memory, computing and energy resources. To enable training PINNs on edge devices, this paper proposes an end-to-end compressed PINN based on Tensor-Train decomposition. In solving a Helmholtz equation, our proposed model significantly outperforms the original PINNs with few parameters and achieves satisfactory prediction with up to 15× overall parameter reduction.

1. Introduction

Physics-informed neural networks (PINNs) are increasingly used to solve a wide range of forward and inverse problems involving partial differential equations (PDEs), including fluids mechanics (Raissi et al., 2020), materials modeling (Liu & Wang, 2019), safety verification (Bansal & Tomlin, 2021) and control (Onken et al., 2021) of autonomous systems. Despite their success of learning complex systems using the simple multilayer perception (MLP) architecture, large neural networks are often required to achieve high expressive power. This has significantly increased the memory and computing cost of training a PINN. Furthermore, a PINN often has to be trained many times in practice once the problem setting (e.g., boundary condition, measurement data, safety specification) changes.

It is increasingly important to enable PINN training on resource-constraint edge devices. On one side, safety-aware learning-based verification and control (Bansal & Tomlin, 2021; Onken et al., 2021) often require the PINN to be trained on a tiny embedded processor of an autonomous agent. On the other side, the emerging digital twin and smart manufacturing need AI-assistant design with IP protection (Stevens et al., 2020), where federated learning with many edge devices allows users to design shared AI models without disclosing their private data. In both cases, training has to be done on edge devices with very limited memory, computing and energy budget.

This paper proposes **TT-PINN**, an end-to-end tensor-compressed method for training PINNs. This method achieves huge parameter and memory reduction in the training process, by combining Tensor-Train compressed model representation and a physics-informed network to approximate the solutions of PDEs. We use this method to solve a Helmholtz equation and compare it with standard PINNs. With only thousands of parameters, our models significantly outperform the original PINNs of similar or larger sizes.

2. Background: PINN

We consider the problem of solving a PDE

$$\begin{aligned} &\boldsymbol{u}_t + \mathcal{N}_{\boldsymbol{x}}[\boldsymbol{u}] = 0, \quad \boldsymbol{x} \in \Omega, t \in [0, T] \\ &\boldsymbol{u}(\boldsymbol{x}, 0) = h(\boldsymbol{x}), \quad x \in \Omega, \\ &\boldsymbol{u}(\boldsymbol{x}, t) = g(\boldsymbol{x}, t), \quad t \in [0, T], \quad \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{1}$$

where $\boldsymbol{x}$ and t are the spatial and temporal coordinates respectively, Ω and $\partial\Omega$ denote the computational domain and its boundary; $\mathcal{N}_{\boldsymbol{x}}$ is a general linear or nonlinear operator; $\boldsymbol{u}(\boldsymbol{x}, t)$ is the solution of the above PDE with the initial condition $h(\boldsymbol{x})$ and the boundary condition. In PINNs (Raissi et al., 2019), a neural network approximation $\boldsymbol{u}(\boldsymbol{x}, t) \approx f_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$ parameterized by $\boldsymbol{\theta}$ is substituted into the PDE (1) and yields a residual defined as

$$\boldsymbol{r}_{\boldsymbol{\theta}}(\boldsymbol{x}, t) := \frac{\partial}{\partial t} f_{\boldsymbol{\theta}}(\boldsymbol{x}, t) + \mathcal{N}_{\boldsymbol{x}}\left[f_{\boldsymbol{\theta}}(\boldsymbol{x}, t)\right]. \tag{2}$$

We train parameters $\boldsymbol{\theta}$ by minimizing the loss function

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_b + \mathcal{L}_0. \tag{3}$$

[*]Equal contribution [1]Department of Statistics and Applied Probability, University of California, Santa Barbara, CA, United States [2]Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, United States. Correspondence to: Ziyue Liu <ziyueliu@ucsb.edu>, Xinling Yu <xyu644@ucsb.edu>, Zheng Zhang <zhengzhang@ece.ucsb.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

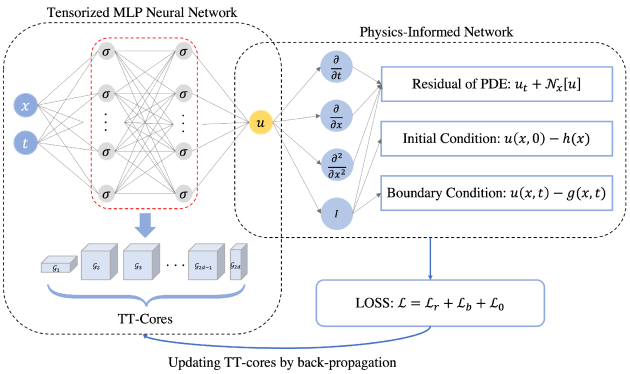

Figure 1. The proposed TT-PINN framework. The Left part is a tensorized MLP neural network with one tensorized hidden layer, in which the trainable parameters are stored as multiple TT-cores. The right part is the embedded governing physical laws that are used to identify the PDE and to force the neural network approximating the solution. During the training, the TT-cores are directly updated by its gradients calculated via auto-differentiation.

Here

$$\mathcal{L}_r = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|\boldsymbol{r}_{\boldsymbol{\theta}}(\boldsymbol{x}_{\boldsymbol{r}}^{\boldsymbol{i}}, t_r^i)\right|^2, \mathcal{L}_b = \frac{1}{N_b}\sum_{i=1}^{N_b}\left|f_{\boldsymbol{\theta}}\left(\mathbf{x}_b^i, t_b^i\right) - g_b^i)\right|^2,$$
$$\mathcal{L}_0 = \frac{1}{N_0}\sum_{i=1}^{N_0}\left|f_{\boldsymbol{\theta}}\left(\mathbf{x}_0^i, 0\right) - h_0^i)\right|^2 \tag{4}$$

penalize the residual of the PDE, the boundary conditions and the initial conditions respectively; N_r, N_b, and N_0 are the numbers of data points for corresponding loss terms.

3. The TT-PINN Method

3.1. TT-PINN Architecture

In this work, we consider tensor-compressed training of PINN based on a multilayer perception (MLP) network. A standard MLP uses an L-layer cascaded function

$$\boldsymbol{z}_k = \sigma\left(\mathbf{W}_k\boldsymbol{z}_{k-1} + \boldsymbol{b}_k\right),\ k = 1, 2, \cdots, L \tag{5}$$

with $\boldsymbol{z}_0 = [\boldsymbol{x}, t]$ and $\boldsymbol{u} = \boldsymbol{z}_L$ to approximate the solution. The weight matrix $\mathbf{W}_k$ can consume lots of memory, making the training unaffordable on edge devices. This challenge becomes more significant when the PDE operator involves highly inhomogeneous material properties or strongly scattered waves. In these cases, large neural networks are often needed to obtain high expressive powers.

As shown in the Figure 1, the TT-PINN replaces the weight matrix of an MLP layer by a series of TT-cores in the training process. For simplicity, we drop the layer index, and let $\mathbf{W} \in \mathbb{R}^{M\times N}$ denote a generic weight matrix in an MLP layer. We factorize its dimension sizes as $M = \prod_{i=1}^{d} m_i$ and $N = \prod_{j=1}^{d} n_j$, fold $\mathbf{W}$ into a $2d$-way tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{m_1\times m_2\times\cdots\times m_d\times n_1\times n_2\times\cdots\times n_d}$, and approximate $\boldsymbol{\mathcal{W}}$ with the TT-decomposition (Oseledets, 2011):

$$\widehat{\boldsymbol{\mathcal{W}}}(i_1, i_2, \ldots, i_d, j_1, j_2, \ldots, j_d) = \mathbf{G}_1(i_1)\ldots\mathbf{G}_d(i_d)\mathbf{G}_{d+1}(j_1)\ldots\mathbf{G}_{2d}(j_d). \tag{6}$$

Here $\mathbf{G}_k(i_k) \in \mathbb{R}^{r_{k-1}\times r_k}$ is the i_k-th slice of the TT-core $\boldsymbol{\mathcal{G}}_k \in \mathbb{R}^{r_{k-1}\times m_k\times r_k}$ by fixing its 2nd index as i_k. The vector $(r_0, r_1, \ldots, r_{2d})$ is called TT-ranks with the constraint $r_0 = r_{2d} = 1$. This TT representation reduces the number of unknown variables in a weight matrix from $\prod_{k=1}^{d} m_k n_k$ to $\sum_{k=1}^{d} r_{k-1}m_k r_k + r_{d+k-1}n_k r_{d+1}$. The compression ratio can be controlled by the TT-ranks. Recent approaches can learn proper TT-ranks automatically in the training process via a Bayesian formulation (Hawkins & Zhang, 2021; Hawkins et al., 2022).

In most existing works of TT-layer (Novikov et al., 2015), a Tensor-Train-Matrix (TTM) decomposition is used, in which the weight matrix is represented by d 4-way TT-cores instead of $2d$ 3-way TT-cores as we described above. Here we adopt TT instead of TTM, because the TT format allows easy tensor-network contraction (shown in Section 3.2), which can greatly reduce the memory and computational cost in both forward and backward propagation.

3.2. Forward & Backward Propagation of TT-PINN

Compared to techniques that compress a well-trained model for inference, TT-PINNs are directly trained in the compressed format. Specifically, the TT-cores that approximate a weight matrix are directly used in the forward propagation and updated in the backward propagation.

Memory-Efficient Forward Propagation. As shown in (5), the main cost in a forward pass is computing a matrix vector product like $\mathbf{W}\boldsymbol{z}$. Instead of reconstructing $\mathbf{W}$ from its TT-cores, we directly use its low-rank TT-cores to obtain the result. Specifically, let $\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{n_1\times n_2\cdots\times n_d}$ be the folding of $\boldsymbol{z}$ into a d-way tensor, then TT-PINN computes a series of tensor-network contractions between tensor $\boldsymbol{\mathcal{Z}}$ and the TT-cores $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{2d}$ as shown in Fig. 2.

We use the tensor-network notation (Orús, 2014; Cichocki, 2014) to show the computation process. A generic N-way tensor is represented by circle and N edges; a shared edge among two tensors mean production (i.e., contraction) along that dimension. Before the computation starts, TT-cores are neither connected to each other nor connected to the tensor $\boldsymbol{\mathcal{Z}}$. We now explain the whole process by three steps. (1) Firstly, the tensor $\boldsymbol{\mathcal{Z}}$ contracts with the last TT-core $\boldsymbol{\mathcal{G}}_{2d}$ as shown by the red dashed rectangle in Fig. 2 (a), producing an intermediate tensor $\boldsymbol{\mathcal{Z}}_1$, in which the size of the d-th

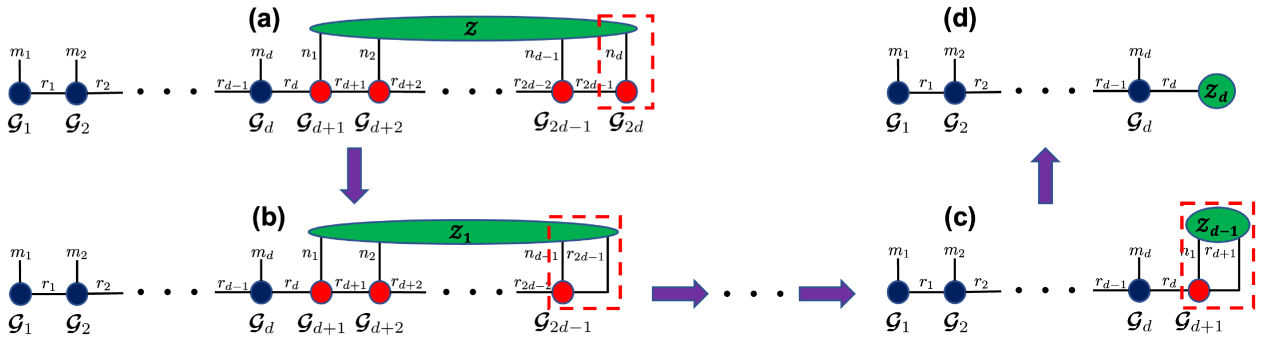

Figure 2. Matrix-vector product in the forward propagation using low-rank TT-cores only.

dimension changes from n_d to r_{2d-1} and all the other dimensions remains unchanged. (2) In the second step, the rest of the red TT-cores are contracted in sequence, from $\mathcal{G}_{2d-1}$ to $\mathcal{G}_{d+1}$. Fig. 2 (b) shows the contraction between the first intermediate tensor $\mathcal{Z}_1$ and $\mathcal{G}_{2d-1}$ on two dimensions, producing a $(d-1)$-way tensor $\mathcal{Z}_2 \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_{d-2} \times r_{2d-2}}$. Similarly, each time the k-th intermediate tensor $\mathcal{Z}_k$ contracts with the $(2d-k)$-th TT-core $\mathcal{G}_{2d-k}$, and the resulting tensor $\mathcal{Z}_{k+1}$ will have one dimension eliminated. After $\mathcal{Z}_{d-1}$ contracts with the last red TT-core $\mathcal{G}_{d+1}$, the resulting tensor $\mathcal{Z}_d$ will only have one dimension of size r_d, as shown in Part (c) and (d) of Fig. 2. (3) Finally, we contract $\mathcal{Z}_d$ with $\mathcal{G}_d$ and connect all the other TT-cores together by sequentially contracting $\mathcal{G}_d$ with $\mathcal{G}_{d-1}$, $\mathcal{G}_{d-2}$, and all the way to $\mathcal{G}_1$, obtaining the final result as a vector of size $m_1 m_2 \cdots m_d$.

Backward Propagation. After the forward propagation, our proposed TT-PINNs calculate the customized loss function similarly as the traditional PINNs, then the backward propagation begins, in which the auto-differentiation (AD) algorithm (Baydin et al., 2018) is applied. Since the AD automatically records each computation step and the evolved objects during the forward pass to generate a so-called computational graph that is used to calculate the gradient of the loss w.r.t each object through the chain rule, we are able to obtain the gradient for each TT-core thus directly updating each TT-core using stochastic gradient descent.

Through the whole process of the forward and backward propagations in the proposed TT-PINNs, all computations are done on the compressed parameters, i.e., TT-cores, instead of a full-size weight matrix. Therefore, this end-to-end compressed training framework can largely reduce the memory cost during the training.

4. Experiments and Results

In this section, we present a series of numerical studies to assess the performance of the proposed TT-PINN against a standard MLP PINN. Specifically, we consider a two-dimensional Helmholtz PDE:

$$\begin{aligned} &(\Delta + k^2)u(x,y) - g(x,y) = 0, \quad (x,y) \in \Omega := [0,1]^2, \\ &u(x,y) = 0, \quad (x,y) \in \partial\Omega, \end{aligned} \tag{7}$$

where Δ is the Laplace operator and $k = 4\pi$ is the wave number. The exact solution to this problem takes the form $u(x,y) = \sin(kx)\sin(ky)$, corresponding to a source term

$$g(x,y) = k^2 \sin(kx)\sin(ky). \tag{8}$$

The PINN approximation $u_{\boldsymbol{\theta}}(x,y)$ to solving (7) can be constructed by parametrizing its solution with a deep neural network $f_{\boldsymbol{\theta}}(x,y)$

$$u_{\boldsymbol{\theta}}(x,y) = x(x-1)y(y-1)f_{\boldsymbol{\theta}}(x,y). \tag{9}$$

The above transformation is applied to the neural network to exactly meet the Dirichlet boundary condition (Lu et al., 2021). Then the parameters $\boldsymbol{\theta}$ can be identified by minimizing the total residual at $N_r = 1200$ collocation points that are randomly placed inside the domain Ω.

We use this benchmark problem to compare the performances of TT-PINNs against PINNs in terms of the total number of parameters. Specifically, we consider a set of neural networks with 3 hidden layers, and we control the number of parameters by varying the number of neurons per layer for PINNs and the choice of TT-ranks for TT-PINNs. In TT-PINNs, the TT-ranks were determined by the desired compression ratio for each hidden layer. For example, to compress a 256×256 weight matrix $\mathbf{W}_h$ in a fully-connected layer with $40\times$ compression, the TT-ranks are determined as $(1,8,8,8,8,8,8,8,1)$ when factorizing each dimension of $\mathbf{W}_h$ as $256 = 4^4$. It is also possible to automatically determine the TT-ranks via the Bayesian tensor rank determination in (Hawkins et al., 2022; Hawkins & Zhang, 2021). To guarantee convergence, all models are trained with 40,000 iterations. As for the training settings, we use the Adam optimizer (Kingma & Ba, 2014) with an initial learning rate 10^{-3} decayed by the factor of 0.9 after each 1000 iterations. The neural networks are initialized by

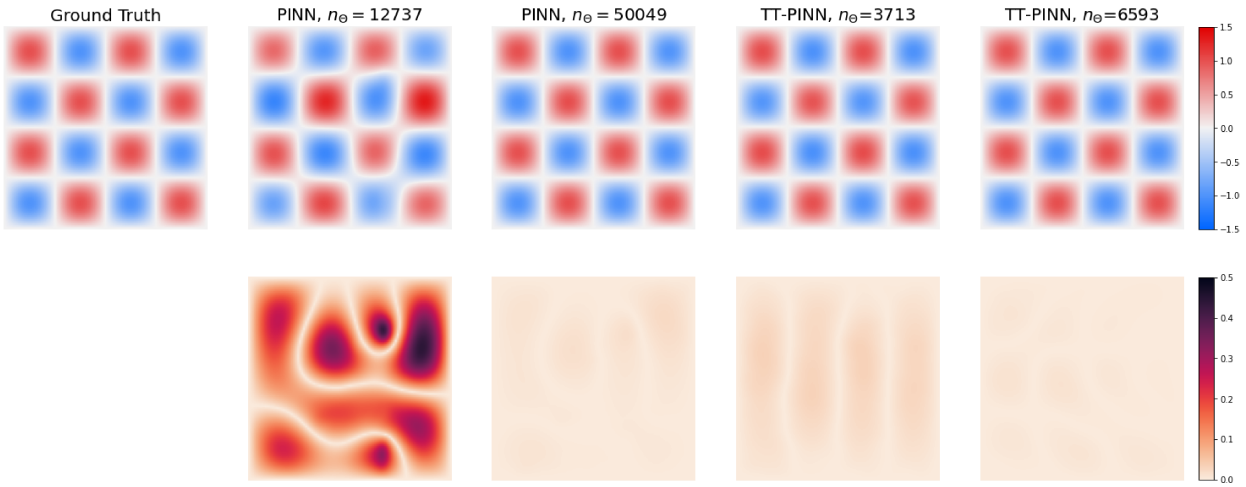

Figure 3. Comparison of PINNs and TT-PINNs in solving the Helmholtz equation (7). The first row contains the prediction results compared to the ground-truth solution. The second row shows the corresponding absolute errors. Our approach achieves similarly accurate prediction while using $15\times$ less parameters than the original PINN.

Table 1. The performance of PINNs in solving Helmholtz equation for different model sizes. Here $\mathbf{W}_h$ represents the weight matrix of each hidden layer and $n_{\boldsymbol{\theta}}$ is the total number of parameters. The mean squared errors and relative ℓ_2 errors are reported.

$\mathbf{W}_h$	$n_{\boldsymbol{\theta}}$	MSE	Rel. ℓ_2 error
32×32	3297	1.32e-1	7.34e-1
64×64	12737	1.56e-2	2.52e-1
128×128	50049	2.60e-5	1.04e-2
256×256	198401	**1.00e-6**	**2.07e-3**

Table 2. The performance of TT-PINNs in solving Helmholtz equation for different model sizes. Here $\widehat{\mathbf{W}}_h$ represents the weight matrix approximated by the TT-cores in each tensorized hidden layer. $n_{\boldsymbol{\theta}}$ is the total number of parameters in the TT-PINN. The mean squared errors and relative ℓ_2 errors are reported.

$\widehat{\mathbf{W}}_h$	Compression	$n_{\boldsymbol{\theta}}$	MSE	Rel. ℓ_2 error
128×128	$40\times$	2169	2.42e-4	3.14e-2
128×128	$20\times$	3597	3.08e-4	3.55e-2
256×256	$100\times$	3713	2.25e-4	3.03e-2
256×256	$40\times$	6593	1.50e-5	7.75e-3
256×256	$20\times$	12449	**4.00e-6**	**4.26e-3**

the Xavier initialization scheme (Glorot & Bengio, 2010), and a Sine activation function is applied to each neuron.

Table 1 and Table 2 summarize our results. Clearly, the expressive power of both standard PINNs and the proposed TT-PINN scales with its model size: larger models provide better approximation to the ground-truth solution. However, our proposed TT-PINNs achieves satisfactory prediction while using much less parameters than a fully connected 3-layer PINN with 256 neurons per layer. To avoid any confusion, the compression ratios reported in Table 2 are for tensorized hidden layers, not for the whole model because so far we only tensorize the hidden layers and leave the input layer and the output layer uncompressed.

Figure 3 shows a visualized comparison of the prediction performance between TT-PINNs and PINNs. As can be seen, the PINN with $n_{\boldsymbol{\theta}} = 12737$ model parameters, which corresponds to 64 neurons per layer, produces the worst prediction among the 4 models. Meanwhile, the proposed TT-PINN with only $n_{\boldsymbol{\theta}} = 3713$ parameters, which corresponds to compressing a 256×256 weight matrix by $100\times$ in the training, achieves a significantly improved prediction. Also, the TT-PINN with $n_{\boldsymbol{\theta}} = 3713$ parameters yields an equally accurate prediction as the PINN with 50049 model parameters. These results show that, by approximating a more complicated neural network with the low rank structure (i.e., TT-cores), our proposed TT-PINNs are capable of, in some level, preserving the expressive power of a larger PINN. This will greatly reduce the requirement of hardware resources in edge computing.

5. Conclusion and Discussions

In this paper, we have proposed an end-to-end compressed architecture for training PINNs with less computing resources. It is the first time that a low-rank structure is applied to achieve memory efficiency while maintaining satisfactory performance in training PINNs. This work is a promising solution for training PINNs on edge devices.

This work, however, is still at the early stage thus very limited. Firstly, the PDE we considered in this work is relatively simple and does not have stiffness issue that frequently occurs in many engineering problems. Secondly, the current

network size we have considered is still relatively small, the performance of TT-PINN needs to be demonstrated on larger PINNs. Finally, deploying this framework on edge computing platforms (e.g., embedded GPU or FPGA) requires further algorithm/hardware co-design.

Acknowledgement

This work was supported by NSF # 1817037 and NSF # 2107321.

References

Bansal, S. and Tomlin, C. J. Deepreach: A deep learning approach to high-dimensional reachability. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1817–1824. IEEE, 2021.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

Cichocki, A. Tensor networks for big data analytics and large-scale optimization problems. *arXiv preprint arXiv:1407.3124*, 2014.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Hawkins, C. and Zhang, Z. Bayesian tensorized neural networks with automatic rank selection. *Neurocomputing*, 453:172–180, 2021.

Hawkins, C., Liu, X., and Zhang, Z. Towards compact neural networks via end-to-end training: A Bayesian tensor approach with automatic rank determination. *SIAM Journal on Mathematics of Data Science*, 4(1):46–71, 2022.

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

Liu, D. and Wang, Y. Multi-fidelity physics-constrained neural network and its application in materials modeling. *Journal of Mechanical Design*, 141(12), 2019.

Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., and Johnson, S. G. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. Tensorizing neural networks. *Advances in neural information processing systems*, 28, 2015.

Onken, D., Nurbekyan, L., Li, X., Fung, S. W., Osher, S., and Ruthotto, L. A neural network approach applied to multi-agent optimal control. In *European Control Conference (ECC)*, pp. 1036–1041, 2021.

Orús, R. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of physics*, 349:117–158, 2014.

Oseledets, I. V. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

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

Raissi, M., Yazdani, A., and Karniadakis, G. E. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

Stevens, R., Taylor, V., Nichols, J., Maccabe, A. B., Yelick, K., and Brown, D. AI for science. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States), 2020.