Accelerating training of Physics Informed Neural Network for 1D PDEs with Hierarchical Matrices

Mateusz Dobija[1,3][0000−0003−4557−3534],
Anna Paszyńska[1][0000−0002−0716−0619],
Carlos Uriarte[4][0000−0002−0155−3091], and
Maciej Paszyński[2][0000−0001−7766−6052]

[1] Jagiellonian University, Krakow, Poland
mateusz.dobija@doctoral.uj.edu.pl
anna.paszynska@uj.edu.pl
[2] AGH University of Krakow, Poland
maciej.paszynski@agh.edu.pl
[3] Doctoral School of Exact and Natural Sciences, Jagiellonian University
[4] Basque Center for Applied Mathematics, Bilbao, Spain curiarte@bcamath.org

Abstract. In this paper, we consider a training of Physics Informed Neural Networks with fully connected neural networks for approximation of solutions of one-dimensional advection-diffusion problem. In this context, the neural network is interpreted as a non-linear function of one spatial variable, approximating the solution scalar field, namely $y = PINN(x) = A_n\sigma(A_{n-1}...A_2\sigma(A_1 + b1) + b2) + ... + b_{n-1}) + b_n$. In the standard PINN approach, the A_i denotes dense matrices, b_i denotes bias vectors, and σ is the non-linear activation function (sigmoid in our case). In our paper, we consider a case when A_i are hierarchical matrices $A_i = \mathcal{H}_i$. We assume a structure of our hierarchical matrices approximating the structure of finite difference matrices employed to solve analogous PDEs. In this sense, we propose a hierarchical neural network for training and approximation of PDEs using the PINN method. We verify our method on the example of a one-dimensional advection-diffusion problem.

Keywords: Partial Differential Equations, Physics Informed Neural Networks, Hierarchical Matrices

1 Introduction

Physics Informed Neural Networks (PINN) was introduced in 2019 by George Karniadakis [17]. PINNs have several applications from fluid mechanics [2, 15, 12, 20, 21], wave propagation [18, 14, 7], phase-filed modeling [8], biomechanics [1, 11], and inverse problems [5, 16, 13]. In this paper, we focus on a one-dimensional

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38

advection-diffusion problem. Its extension to a two-dimensional problem, known as the Eriksson-Johnson model problem [6] can be a subject of our future work. Both one and two-dimensional model problems are often employed for testing the convergence of finite element method solvers [4, 3]. Following the idea of PINN, we represent the solution of a one-dimensional advection-diffusion problem as the neural network. In order to speed up the training process, the neural network layers are represented by hierarchically compressed matrices.

1.1 One-dimensional advection-diffusion problem

One-dimensional advection-diffusion problem can be defined as:
Find $u \in C^2(0,1)$:

$$\underbrace{-\epsilon \frac{d^2u(x)}{dx^2}}_{\text{diffusion}=\epsilon} \underbrace{+\beta \frac{du(x)}{dx}}_{\text{advection ``wind''}=1} = 0, x \in (0,1). \tag{1}$$

The problem is augmented with boundary conditions

$$-\epsilon \frac{du}{dx}(0) + u(0) = 1.0, \; u(1) = 0. \tag{2}$$

The solution of this problem has the boundary layer of thickness ϵ at the right corner of the domain, as it is illustrated in Figure 1.

1.2 PINN for one-dimensional advection-diffusion problem.

Following the idea of PINN, we represent the solution as the neural network:

$$u(x) = PINN(x) = A_n \sigma \left(A_{n-1} \sigma (\dots \sigma (A_1 x + B_1) \dots + B_{n-1}) + B_n \right. \tag{3}$$

We define the loss function for the residual of the PDE

$$LOSS_{PDE}(x) = \left(-\epsilon \frac{d^2 PINN(x)}{dx^2} - \beta \frac{dPINN(x)}{dx} - 1 \right)^2 \tag{4}$$

We also define the loss function for the left boundary condition

$$LOSS_{BC0} = \left(-\epsilon \frac{dPINN}{dx}(0) + PINN(0) - 1.0 \right)^2, \tag{5}$$

and the loss function for the right boundary condition

$$LOSS_{BC1} = \left(PINN(1) \right)^2, \tag{6}$$

The total loss function is defined by combining a weighted sum

$$LOSS = w_{PDE} \sum_{x \in (0,1)} \left(LOSS_{PDE}(x) \right)^2 \tag{7}$$

$$+ w_{BC0} \left(LOSS_{BC0}(0) \right)^2 \tag{8}$$

$$+ w_{BC1} \left(LOSS_{BC1}(1) \right)^2. \tag{9}$$

The PINN methods can succesfully solve the advection-diffusion problem as it is shown in [19].

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38

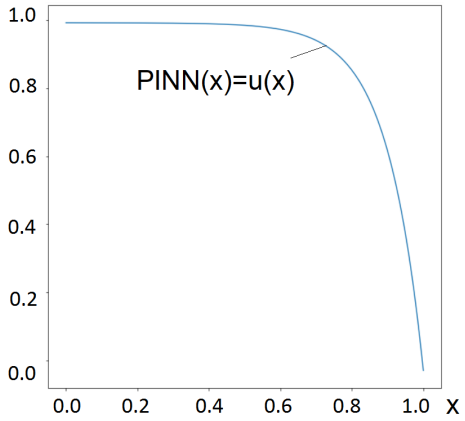


Fig. 1: PINN solution of the advection-diffusion problem for $\epsilon = 0.1$.

1.3 Neural network with hierarchical matrices.

In this paper, we investigate the fully connected neural networks with hierarchical matrices [10, 9], see Figure 2, namely $y = PINN(x) = \mathcal{H}_n\sigma(\mathcal{H}_{n-1}...\mathcal{H}_2\sigma(\mathcal{H}_1 + b1) + b2) + ... + b_{n-1}) + b_n$, where $\mathcal{H}_i$ are hierarchical matrices, and b_i are normal vectors. We do not compress the dense matrices of layers from the traditional fully connected neural network. We rather assume the structure of the compressed matrix for a layer, and we train it from the very beginning in the compressed form. We implemented our own training kernel for the matrices of layers compressed in a hierarchical manner. We have assumed that each layer has a structure of hierarchical matrix of rank one. The matrix is "refined" towards the diagonal, and the off-diagonal blocks are rank 1. In other words, each off-diagonal block is represented as a multiplication of 1 column and 1 row.

The main benefit of hierarchical matrices is that they enable linear computational cost matrix-vector multiplication, see Figure 4.

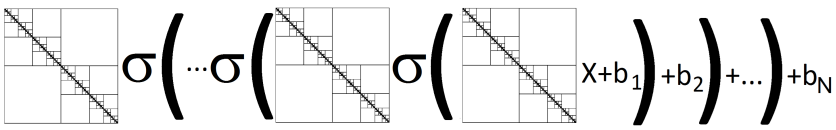


Fig. 2: Neural network with hierarchical matrices.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38

2 Matrix compression

The main idea of speeding up the training process is storing the weight matrices A_i in a compressed form. The matrix compression used in the paper is based on Recursive Singular Value Decomposition, which is the key idea behind the hierarchical matrices [9, 10]. In the Recursive Singular Value Decomposition Compression algorithm, the matrix is recursively divided into four smaller submatrices, and selected submatrices (for which so-called admissibility condition is fulfilled) are approximated using the singular value decomposition algorithm (SVD). The remaining submatrices, for which the admissiblity condition is not fulfilled, are recursively divided into smaller matrices. The SVD algorithm decomposes matrix A into three matrices $A = UDV$, where D is diagonal matrix with singular values sorted in descending order, U is the matrix of columns, and V is the matrix of rows. In the approximate SVD, the singular values smaller than the predefined threshold are removed, together with the corresponding rows from V and columns from U.

The compressed matrix can be stored in a tree-like structure, where the root node corresponds to the whole matrix, each node can have four sons corresponding to submatrices of the matrix or can be a leaf representing the corresponding matrix in the SVD compressed form. Figure 3 shows an exemplary tree representing the hierarchically compressed matrix.

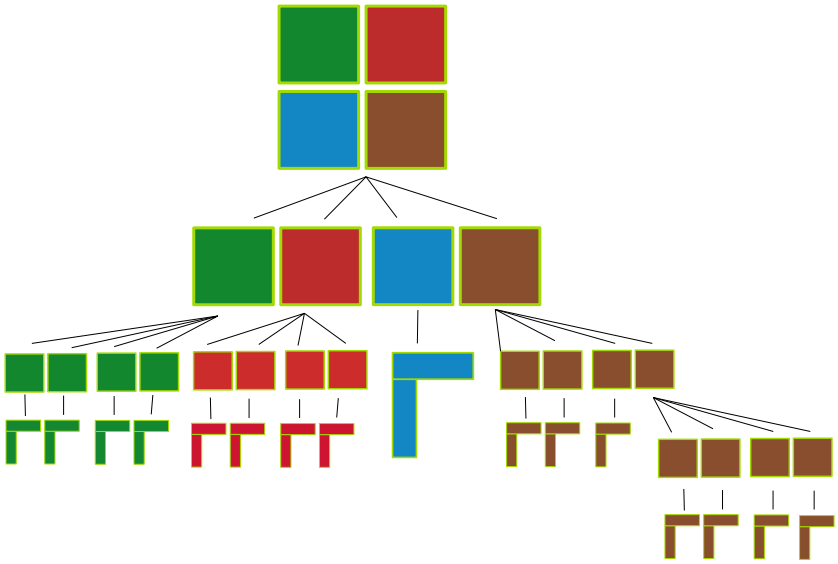


Fig. 3: An exemplary tree representing the hierarchically compressed matrix.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38

3 The compressed matrix-vector multiplication

The SVD compressed submatrices are stored as multi-columns U multiplied by multi-rows (DV). The SVD compression of a matrix allows to speed up the matrix-vector multiplication algorithm - the time complexity is $(O(Nr))$, where N is the size of the uncompressed matrix and r is the number of singular values bigger than a given threshold (rank of the matrix). Figure 4 presents the idea of SVD compressed matrix-vector multiplication.

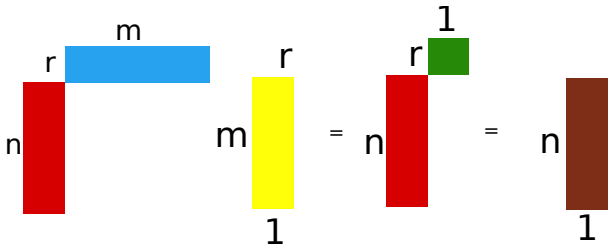


Fig. 4: The idea of SVD compressed matrix by vector multiplication.

The recursive algorithm for multiplication of the hierarchically compressed matrix by a vector takes as input a node representing the tree (hierarchically compressed matrix) or a node representing part of the hierarchically compressed matrix, and a vector. The algorithm works recursively. If the input node has no children, it represents the SVD compressed part of the matrix (stored as multi-columns U and multi-rows DV (result of multiplication of diagonal matrix D by V). In such a case, the result is calculated as the multiplication $U((DV) * v)$. It must be underlined that the order of performing multiplication is important because it hardly influences the computational cost of calculating the results. If the input node has children, the partial multiplication for each child (submatrix) by the corresponding part of vector v is performed recursively, then the final result of multiplications of children is calculated. The algorithm is presented in Algorithm 1.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38

Algorithm 1 MultiplyMatrixByVector

Require: node T, vector to multiply v
  if $T.sons = \emptyset$ **then**
    return $T.U * (T.V * v)$;
  end if
  $numRows$ =number of rows of vector v;
  $v_1 = v(1 : floor(numRows/2), :)$ //first part of vector v
  $v2 = v(floor(numRows/2 + 1) : numRows, :)$ //second part of vector v
  res1=MultiplyMatrixByVector(T.children(1),v1)
  res2=MultiplyMatrixByVector(T.children(2),v2)
  res3=MultiplyMatrixByVector(T.children(3),v1)
  res4=MultiplyMatrixByVector(T.children(4),v2)
  //calculate the final result of multiplication
  res1res2=res1+res2
  res3res4=res3+res4
  return result=[res1res2;res3res4]

4 Using the algorithm of hierarchically compressed matrix-vector multiplication to speed up neural network training

The main idea is to store the weight matrix A in the hierarchically compressed form. The matrix of size $n \times n$ can be represented as the hierarchically compressed matrix of rank 1, where on each level of hierarchy the off-diagonal blocks are represented by SVD compressed blocks, and the remaining blocks are divided into smaller blocks. On this lower level of the compression, again, the off-diagonal blocks are represented by SVD compressed blocks, and the remaining blocks are divided into smaller blocks. The process of "refinement" of the matrix is stopped if the if the submatrix has size 1. An exemplary hierarchical compressed matrix of size 8×8 is presented in figure 5. The number of entries of the compressed form of the matrix of size $n \times n$ is equal to $2 * n * log_2(2 * n)$. In our neural network the compressed matrix of weights is represented as the vector of size $2 * n * log_2(2 * n)$, where the first entries are entries corresponding to the second submatrix, then entries for third submatrix, then entries for the first and finally entries for the fourth submatrix. The numbering of submatrices of the matrix as well as the vector form of hierarchical compressed matrix of size $8x8$ is presented in figure 5. In our neural network the iterative version of matrix-vector multiplication algorithm (see algorithm 1), where the $n \times n$ matrix is stored in form of a vector of size $2 * n * log_2(2 * n)$ was used.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38

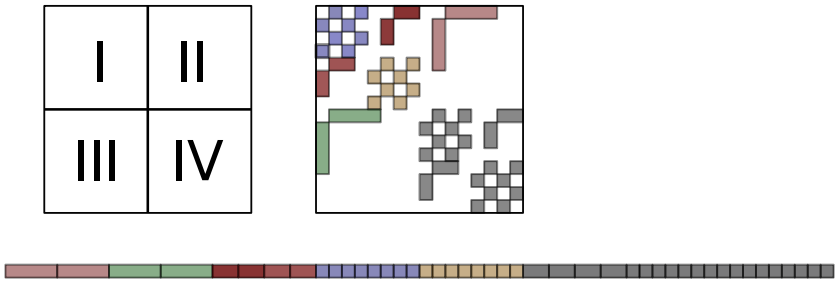


Fig. 5: Compressed matrix and its vector representation.

5 Results

Our test shows that an acceptable solution of the problem defined in equations (1) and (2) was obtained for the $LOSS$ of the order of 0.001. The tests were performed with a learning rate 0.02 and a number of epochs equal to 1000. We have used a neural network with 2 internal layers. In our tests in the classical approach (uncompressed full matrix) each internal layer has number of neurons equal to: $sizeofmatrix$ = 32, 64, 128, 256, 512. So, the number of entries in uncompressed matrices were equal to: $32^2 = 1024$, $64^2 = 4096$, $128^2 = 16384$, $256^2 = 65536$, $512^2 = 262144$.

The number of entries in compressed matrices, corresponding to uncompressed matrices of sizes 32^2, 64^2, 128^2, 256^2, 512^2 where equal to: 384, 896, 2048, 4608, 10240. The convergence of training is presented in Figures 6a-9a for uncompressed matrix with classic matrix-vector multiplication. For the fully connected neural network with full matrices and classical matrix-vector multiplication algorithm, the training finds correct solutions for matrices of sizes 32^2, 64^2, and 128^2. The final loss was of order 0.001, and the shape of solution was correct. For example, Figure 6a presents the successful training of neural network with two layers represented by full matrices of size 32^2, where the loss function goes below 0.001.

However, the training does not find a correct solution after 2000 epochs for matrices of size 256^2 and 512^2 (the final loss was higher than 0.001 and the solution shape was wrong). For example Figure 9a presents the training for neural network with two layers represented by full matrices of size 256^2.

The convergence of training for compressed matrix with compressed matrix-vector multiplication is presented in Figures 6b-9b. For a neural network with 2 layers with hierarchical matrices corresponding to matrices of size 32^2, with total of 384 non-zero entries to train, the convergence of training is presented in Figure 6b. The loss value reaches 10^{-3} after 600 epochs. This accuracy and convergence rate is similar to the classical training presented in Figure 6a. On top of that, the matrix-vector multiplication is way cheaper in the compressed matrix NN. Comparing Table 1 and Table 2, we can see that compressed matrices can be

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38

trained 3 times faster for this smaller neural network (see first rows in Table 1 and Table 2.

For a neural network with 2 layers with hierarchical matrices corresponding to matrices of size 64^2, the convergence of training is presented in Figure 7b. The loss value reaches 10^{-3} after 600 epochs. The compressed matrix-vector multiplications are cheaper, and the compressed neural network can be trained 2 times faster (see second row in Table 1 and Table 2).

Finally, for a neural network with 2 layers with hierarchical matrices corresponding to matrices of size 128^2, the convergence of training is presented in Figure 8b. The loss value reaches 10^{-3} after 600 epochs. The compressed matrix-vector multiplications are cheaper, and the compressed neural network can be trained 5 times faster (see third row in Table 1 and Table 2).

For larger matrices, contrary to the classical NN, we can still train the compressed NN, using 256 or 512 neurons per layer, see Figure 9b. The compressed matrices have a lower order of trainable entries. Thus, it is possible to train a large compressed matrix even if the standard dense approach has not converged yet.

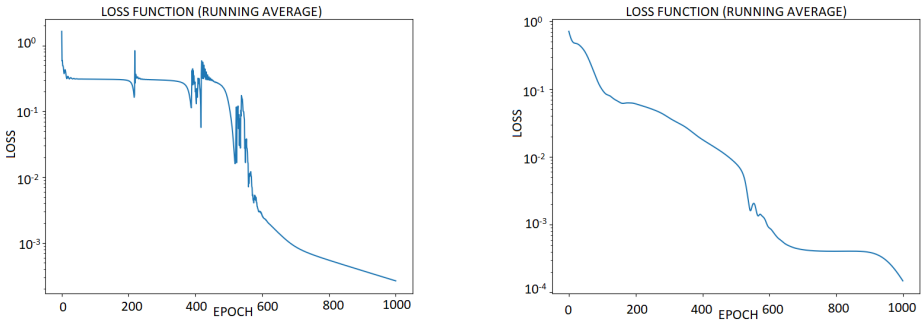
Table 1: Number of epochs and number of FLOPs of classic multiplication, learning rate 0.02, LOSS 0.001

matrix size	number of FLOPs - classic multiplication	number of epochs - classic multiplication
32	620	257,761,280
64	252	419,069,952
128	246	1,629,716,480
256	-	-
512	-	-

Table 2: Number of epochs and number of FLOPs of hierarchical multiplication, learning rate 0.02, LOSS 0.001

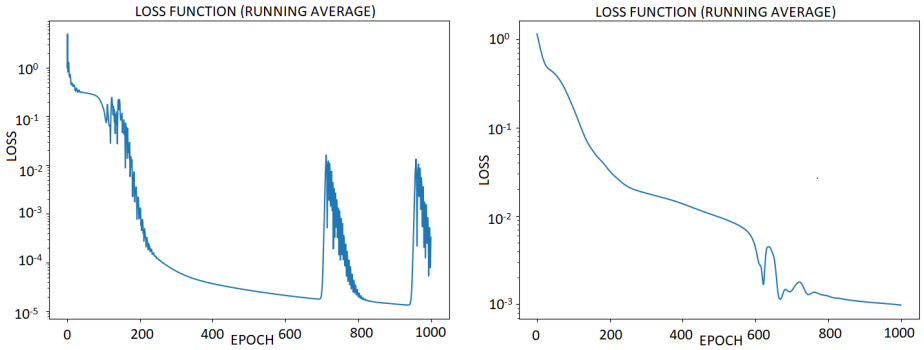
matrix size	number of epochs hierarchical multiplication	number of FLOOPs hierarchical multiplication
32	539	77,172,480
64	658	222,604,928
128	427	333,634,560
256	836	1,478,905,344
512	382	1,512,684,544

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38



(a) (b)

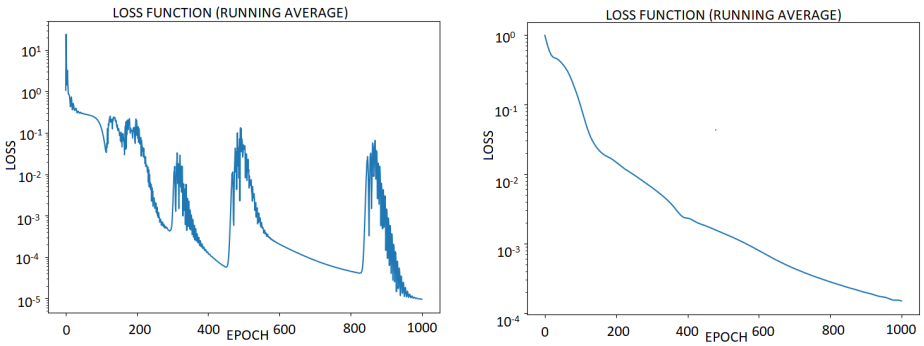
Fig. 6: Left panel: convergence of training of the fully connected neural network with 2 layers, 32 neurons per layer. Right panel: convergence of training of the fully connected neural network with 2 layers, 32 neurons per layer using compressed matrix.



(a) (b)

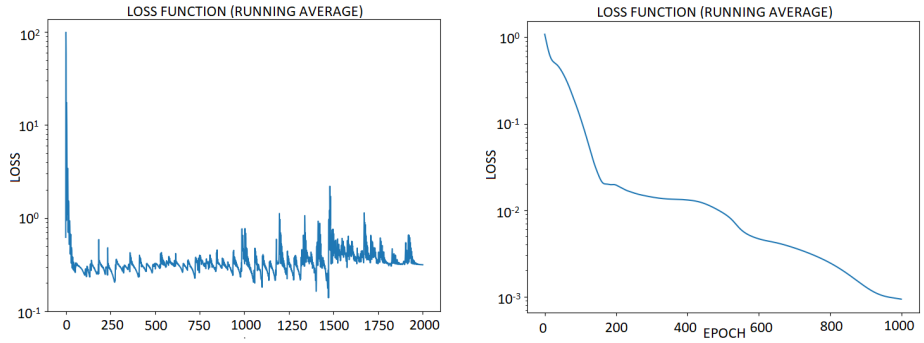
Fig. 7: Left panel: convergence of training of the fully connected neural network with 2 layers, 64 neurons per layer. Right panel: convergence of training of the fully connected neural network with 2 layers, 64 neurons per layer using compressed matrix.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38



(a) (b)

Fig. 8: Left panel: convergence of training of the fully connected neural network with 2 layers, 128 neurons per layer. Right panel: convergence of training of the fully connected neural network with 2 layers, 128 neurons per layer using compressed matrix.



(a) (b)

Fig. 9: Left panel: convergence of training of the fully connected neural network with 2 layers, 256 neurons per layer. Right panel: convergence of training of the fully connected neural network with 2 layers, 256 neurons per layer using compressed matrix.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38

6 Conclusions

In this paper, we proposed a Physics Informed Neural Network with hierarchical matrices for approximation of one-dimensional advection-diffusion problems. The neural network represented a solution of one-dimensional PDE, namely $y = PINN(x) = \mathcal{H}_n\sigma(\mathcal{H}_{n-1}...\mathcal{H}_2\sigma(\mathcal{H}_1 + b1) + b2) + ... + b_{n-1}) + b_n$, where $\mathcal{H}$ are the hierarchical matrices. We have verified our method and showed that it allows to speed up the training process between 2-5 times (compare rows in Tables 1 and 2), while reducing the memory storage up to 3-20 times. The future work will involve generalization of this method for more complex PDEs.

References

1. Alber, M., Tepole, A.B., Cannon, W.R., De, S., Dura-Bernal, S., Garikipati, K., Karniadakis, G., Lytton, W.W., Perdikaris, P., Petzold, L., Kuhl, E.: Integrating machine learning and multiscale modeling-perspectives, challenges, and opportunities in the biologica biomedical, and behavioral sciences. NPJ Digital Medicine **2** (2019). https://doi.org/10.1038/s41746-019-0193-y
2. Cai, S., Mao, Z., Wang, Z., Yin, M., Karniadakis, G.E.: Physics-informed neural networks (PINNs) for fluid mechanics: A review. Acta Mechanica Sinica **37**(12), 1727–1738 (2021)
3. Calo, V., Łoś, M., Deng, Q., Muga, I., Paszyński, M.: Isogeometric Residual Minimization Method (iGRM) with direction splitting preconditioner for stationary advection-dominated diffusion problems. Computer Methods in Applied Mechanics and Engineering **373**, 113214 (2021). https://doi.org/https://doi.org/10.1016/j.cma.2020.113214
4. Chan, J., Evans, J.: A Minimal-Residual Finite Element Method for the Convection–Diffusion Equations. ICES-REPORT **13**(12) (2013), https://oden.utexas.edu/media/reports/2013/1312.pdf
5. Chen, Y., Lu, L., Karniadakis, G.E., Dal Negro, L.: Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Optics express **28**(8), 11618–11633 (2020)
6. Eriksson, K., Johnson, C.: Adaptive finite element methods for parabolic problems I: A linear model problem. SIAM Journal on Numerical Analysis **28**(1), 43–77 (1991), http://www.jstor.org/stable/2157933
7. Geneva, N., Zabaras, N.: Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. Journal of Computational Physics **403** (2020). https://doi.org/10.1016/j.jcp.2019.109056
8. Goswami, S., Anitescu, C., Chakraborty, S., Rabczuk, T.: Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. Theoretical and applied fracture machanics **106** (2020). https://doi.org/10.1016/j.tafmec.2019.102447
9. Hackbusch, W.: A sparse matrix arithmetic based on h-matrices. part i: introduction to h -matrices. Computing **62**, 89–108 (1999)
10. Hackbusch, W.: Hierarchical matrices: algorithms and analysis, vol. 49. Springer (2015)
11. Kissas, G., Yang, Y., Hwuang, E., Witschey, W.R., Detre, J.A., Perdikaris, P.: Machine learning in cardiovascular flows modeling: Predicting arterial blood

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38

pressure from non-invasive 4d flow mri data using physics-informed neural networks. Computer Methods in Applied Mechanics and Engineering **358** (2020). https://doi.org/10.1016/j.cma.2019.112623
12. Ling, J., Kurzawski, A., Templeton, J.: Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. Journal of Fuild Mechanics **807**, 155–166 (2016). https://doi.org/10.1017/jfm.2016.615
13. Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., Johnson, S.G.: Physics-informed neural networks with hard constraints for inverse design. SIAM Journal on Scientific Computing **43**(6), B1105–B1132 (2021). https://doi.org/10.1137/21M1397908
14. Maczuga, P., Paszyński, M.: Influence of activation functions on the convergence of physics-informed neural networks for 1d wave equation. In: Mikyška, J., de Mulatier, C., Paszynski, M., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M. (eds.) Computational Science – ICCS 2023. pp. 74–88. Springer Nature Switzerland, Cham (2023)
15. Mao, Z., Jagtap, A.D., Karniadakis, G.E.: Physics-informed neural networks for high-speed flows. Computer Methods in Applied Mechanics and Engineering **360**, 112789 (2020)
16. Mishra, S., Molinaro, R.: Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs. IMA Journal of Numerical Analysis **42**(2), 981–1022 (2022)
17. Raissi, M., Perdikaris, P., Karniadakis, G.: 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 (Feb 2019). https://doi.org/10.1016/j.jcp.2018.10.045
18. Rasht-Behesht, M., Huber, C., Shukla, K., Karniadakis, G.E.: Physics-informed neural networks (pinns) for wave propagation and full waveform inversions. Journal of Geophysical Research: Solid Earth **127**(5), e2021JB023120 (2022)
19. Sikora, M., Krukowski, P., Paszynska, A., Paszynski, M.: Physics informed neural networks with strong and weak residuals for advection-dominated diffusion problems (2023)
20. Sun, L., Gao, H., Pan, S., Wang, J.X.: Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. Computer Methods in Applied Mechanics and Engineering **361** (2020). https://doi.org/10.1016/j.cma.2019.112732
21. Wandel, N., Weinmann, M., Neidlin, M., Klein, R.: Spline-pinn: Approaching pdes without data using fast, physics-informed hermite-spline cnns. Proceedings of the AAAI Conference on Artificial Intelligence **36**(8), 8529–8538 (2022). https://doi.org/10.1609/aaai.v36i8.20830, https://ojs.aaai.org/index.php/AAAI/article/view/20830

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_38