Neural Networks to solve Partial Differential Equations: a Comparison with Finite Elements

Andrea Sacchetti, Benjamin Bachmann, Kaspar Löffel
School of Engineering - University of Applied Sciences and Arts Northwestern Switzerland

Urs-Martin Künzi, Beatrice Paoli
Laboratory of Web Sciences - Fernfachhochschule Schweiz

We compare the Finite Element Method (FEM) simulation of a standard Partial Differential Equation thermal problem of a plate with a hole with a Neural Network (NN) simulation. The largest deviation from the true solution obtained from FEM (0.025 for a solution on the order of unity) is easily achieved with NN too without much tuning of the hyperparameters. A higher accuracy value (0.001) instead requires refinement with an alternative optimizer to reach a similar performance with NN. A rough comparison between the Floating Point Operations values, as a machine-independent quantification of the computational performance, suggests a significant difference between FEM and NN in favour of the former. This also strongly holds for computation time: for an accuracy on the order of 10^{-5}, FEM and NN require 38 and 1100 seconds, respectively. A detailed analysis of the effect of varying different hyperparameters shows that accuracy and computational time only weakly depend on the major part of them. Accuracies below 0.01 cannot be achieved with the "adam" optimizers and it looks as though accuracies below 10^{-5} cannot be achieved at all. Training turns to be equally effective when performed on points extracted from the FEM mesh.

I. INTRODUCTION

Partial Differential Equations (PDE) govern the behaviour of most physical systems as shown by prominent examples like the Maxwell, diffusion, Navier- Stokes, elasticity, and heat equations. In most practical applications, analytical solutions are not available and one must resort to numerical methods. These include, but are not limited to, finite-element method (FEM), finite differences, gradient discretisation, finite volume, and method of lines. These standard techniques are employed in many commercial packages for physical simulations with FEM being widely the most commonly used. The use of such packages to predict products' behaviour and optimize designs, among other things, has become very much standard in modern industry. The major part of the mentioned methods exploit some sort of discretisation of the considered domain with a network relating the discretisation points to each other, i.e. a *mesh*. In applications where material motion is involved, i.e. in solid and fluid mechanics, it can be advantageous to employ so-called mesh-free methods. Many such techniques have been developed since the introduction of the smoothed particles hydrodynamics in 1977 [1] but they seem to stay confined in somewhat a niche compared to more widespread tools like FEM.

A somewhat revolutionary mesh-free approach was proposed in 1998 [2] which relies on neural networks (NN). The main idea behind it is to exploit two fundamental properties of NNs:

- NNs satisfy the universal approximation theorem, i.e. any given continuous function $\mathbb{R}^n \to \mathbb{R}^m$ can be approximated to arbitrary accuracy with a single-hidden-layer NN of sufficient width
- NNs are infintely-differentiable functions whose derivative can be computed analytically by means of automatic differentiation [3].

These two features make NNs an ideal candidate for a approximating the solution of a given PDE on a mesh-less geometry. The *ansatz* consists in training the network by means of minimising a loss function that is given by the PDE itself computed on a discrete set of points from the considered domain. This is made possible by the fact that the derivatives of the NN are available analytically (instead of numerically) and thus can be computed locally. Boundary conditions, both including the function value or its derivatives, can be accommodated as well, as additional explicit penalties in the loss function.

Since this pioneering work [2], the concept has not been pursued much, mostly due to the fact that NNs were a niche discipline until the late XX century and the required computational power to exploit their capabilities was often not available. The advent of low-cost high-performance hardware as well as the emergence of standardised machine-learning packages including user-friendly NN-libraries (e.g. scikit-learn [4], tensorflow by Google [5], pytorch by Facebook [6]) powered a recent revival of this concept with the appearance of several papers [7–16] and also a few open-source projects [17–20]. It is worth to mention at this stage that many of these studies have a common denominator in their motivation, namely that the NN-approach to solve PDEs can offer several advantages over established techniques like FEM in specific contexts:

1. While the computational power required to train a NN can be significantly larger than that needed to solve a FEM problem, the need for a re-mesh upon change of the underlying geometry can give

name	Reference	back-end	boundaries	dimensions	geometry
DeepXDE*	[17]	T, P	D, N, C	any	CSG
PyDEns	[18]	T	D	any	N
NeuroDiffEq+	[19]	P	D, N	max. 2	B
IDRLnet%	[20]	P	D, N	any	CSG

TABLE I: Comparison of different open-source PDE-NN available packages. Legend - Back-end: T = tensorflow, P = PyTorch; Boundaries: D = Dirichlet, N = von Neumann, C = Custom; Geometry: CSG = Constructive Solid Geometry, B = Basic (simple shapes only), N = Not implemented. Notes: * = support for PyTorch still buggy, + = support for higher dimension coming soon, % = equation types limited.

the upper hand to the NN approach in those cases where the geometry is not fixed.

2. The training of a NN can be improved iteratively: if the accuracy of the obtained solution is not satisfactory in a given range, it can be improved by means of an incremental training with additional points added in the inaccurate region.

3. Thanks to the universal approximation theorem, NNs are generally known to have a strong generalisation power, i.e. the trained solution can accurately predict the true function on points it never came across during training. The standard procedure of training and testing allows to accurately quantify the generalisation power of the obtained solution.

4. The absence of a mesh shall make the NN-approach more convenient as the dimensionality of the problem increases. This applies in particular for 3D-problems and/or time-dependent ones. Very high-dimensionality problems like the multi-particles Schrödinger equation might particularly profit from this aspect.

It is quite surprising though that none of the aforementioned studies provide a systematic, quantitative comparison between conventional numerical solution of a PDE (e.g., via FEM) versus solution via NN in order to prove empirically the existence of said advantages and pin down the conditions under which they occur. In this study, we try to make a first effort to fill this gap by comparing solutions of the heat equation via FEM and a NN.

II. MATERIALS AND METHODS

As a first step, we selected the packages we are using in our study. For FEM, it does not really matter which among the numerous available commercial packages one selects as they are all mature and extremely optimized. The only concern about it, is that this fact might skew the comparison somewhat in favour of FEM because the available packages for PDE-NN are all in their alpha development-stage. This point must be taken into account in the overall comparison. For FEM, we used therefore the standard tool in our team which is the well-established Abaqus package [21]. For the NN-based solution of PDEs we compared four packages that can be found in tab. I:

The package "DeepXDE" currently has the clear upper hand in our analysis: it supports any type of equation in unlimited dimensions with arbitrary boundary conditions, it comes with a powerful built-in engine for constructing the underlying geometry, and it has many additional features and options. Moreover, the developers are very active and constantly improving it while a small community is growing around it. The other packages all miss the one or the other of these features and it was decided not to consider them at this stage. Their development shall nevertheless be monitored carefully in case one of them leaps forward significantly or DeepXDE comes to a halt. In particular "IDRLnet" looks like a possible contender but it is yet in its early development stages and it still lags behind "DeepXDE" at the moment.

In order to compare the PDE-NN approach with FEM we need a relevant sample problem to solve with both methods. Ideally, this shall be a problem admitting an analytical solution and must possess sufficient complexity to provide a reliable test but at the same time it may not require too large of a computational power so that repeated testing is possible. We identified to this purpose a standard example in introductory FEM theory, namely the stationary heat propagation on a squared two-dimensional plate with a circular hole in the centre as in example 8.4 from [22]. The problem reads as follows:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{2a}{\sqrt{x^2+y^2}} - 4 = 0$$

The choice of the non-homogenous term ensures that the exact analytical solution is known and equals

$$T(x,y) = (\sqrt{x^2+y^2} - a)^2.$$

The boundary problem is defined on the squared box of side length $2b$ with a circular hole in the middle of radius a. The boundary conditions are of the Dirichlet (Γ_T) or von Neumann type (Γ_q) as shown in fig. 1. The mathematical formulation of the boundary conditions thus reads:

$$T(r = a) = 0$$

Run	Element order	Nodes	Computation time	Solution Deviation	FLOPs
1	linear	115	11	1.476×10^{-2}	2.26×10^{4}
2	quadratic	327	11	1.625×10^{-3}	1.73×10^{5}
3	linear	172	11	9.128×10^{-3}	3.84×10^{4}
4	quadratic	494	11	8.720×10^{-4}	3.86×10^{5}
5	linear	505	11	3.346×10^{-3}	1.85×10^{5}
6	quadratic	1476	11	3.613×10^{-4}	1.72×10^{6}
7	linear	1847	11.5	1.199×10^{-3}	1.33×10^{6}
8	quadratic	5475	11.5	2.51×10^{-4}	1.19×10^{7}
9	linear	46981	36	2.255×10^{-4}	1.65×10^{8}
10	quadratic	141230	54	5.499×10^{-5}	1.41×10^{9}

TABLE II: Parameters and results of the reference FEM analysis. "Computation time" includes both meshing and solving. FLOPs (floating point operations) only refers to solving.

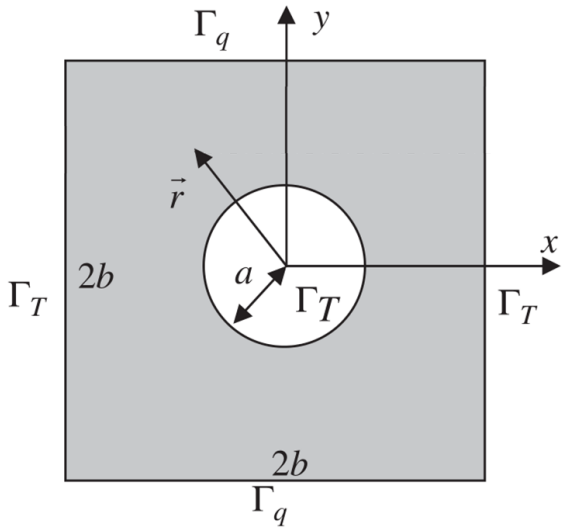


FIG. 1: Domain and boundaries of the heat-propagation problem defined on a squared plate with a hole [22].

$$T(x = \pm b, y) = (\sqrt{b^2 + y^2} - a)^2$$

$$\frac{\partial T}{\partial y}(x, y = \pm b) = \mp 2b \left(\frac{a}{\sqrt{x^2 + b^2}} - 1 \right)$$

As it can be noted, both Dirichlet and von Neumann conditions are defined.

III. RESULTS AND DISCUSSION

As a starting reference point, we take the FEM simulations carried out with Abaqus on the same problem and try to reproduce the performance with NN approach. In the following, we shall always use $a = 0.2$ and $b = 1$.

The FEM simulations have been carried out with five different meshes with varying levels of resolution. In addition, linear and quadratic elements have been used. One example for a relatively coarse mesh (FEM run 1) is represented in fig. 2. The results of these simulations are summarized in tab. II. The deviation with respect to the exact solution is root-mean-square averaged on a 50×50 grid of points within the domain and the number of Floating Point Operations (FLOPs) is obtained directly from Abaqus. We focus in particular on runs 1 and 10 because these represent the extremal points of the convergence analysis, namely they are the simplest and the most accurate simulations, respectively.

We then proceeded to implement the same problem within the "DeepXDE" package and solve it with the aim of achieving a similar accuracy as with FEM. After several test-runs we found hyperparameter-sets that closely reproduce the accuracy of the aformentioned FEM simulations in run 1. These include a choice of the number of training points comparable with the number of FEM elements in run 1 as well as a similar distribution between boundary and domain. The remaining parameters were optimized for accuracy in training, test, and verification on the grid.

The results of the simulations 1-5 are shown in the tab. III. The simulation was repeated 5 times to check the reproducibility of the results. It is apparent that the NN-simulations slightly outperforms FEM in terms of accuracy on a grid of points (column "sol. deviat."). It is also worth to notice that this happens despite the fact that the test error (column "test err.") is systematically larger that the deviation on the grid. This can be traced back to the fact that these two types of errors are indeed not directly comparable: the test error quantifies how much the condition given by the PDE deviates from 0 averaged over the test points and it is in principle not a straightforward measurement of how accurately the NN-solution approximates the true one.

We can also notice a significant discrepancy between the estimated number of FLOPs for the NN-training compared with the FEM-solver, despite of the fact that the computational times are comparable. Part of this discrepancy could be possibly ascribed to the differences between the machines where the simulations have been ran. Moreover the FLOPs estimate for the NN-simulations is affected by several possible sources of error as it merely estimates the FLOPs for a single model prediction and

run	training pts domain	training pts boundary	pts test	intern layers	epochs	optimizer	lern. rate
1-5	50	40	90	20 × 1	1000	adam	1.00e-02

run	test err.	sol. deviat.	compil. time	train. time	FLOPs
1	5.13e-02	2.01e-02	0.7 s	1.2 s	1.54e+08
2	5.61e-02	1.68e-02	0.7 s	1.0 s	1.52e+08
3	4.70e-02	2.54e-02	0.8 s	1.4 s	1.50e+08
4	5.54e-02	1.72e-02	0.6 s	0.9 s	1.45e+08
5	5.09e-02	1.76e-02	0.6 s	0.9 s	1.47e+08

TABLE III: Hyperparameters of the first simulation block (top panel) and corresponding results (bottom panel).

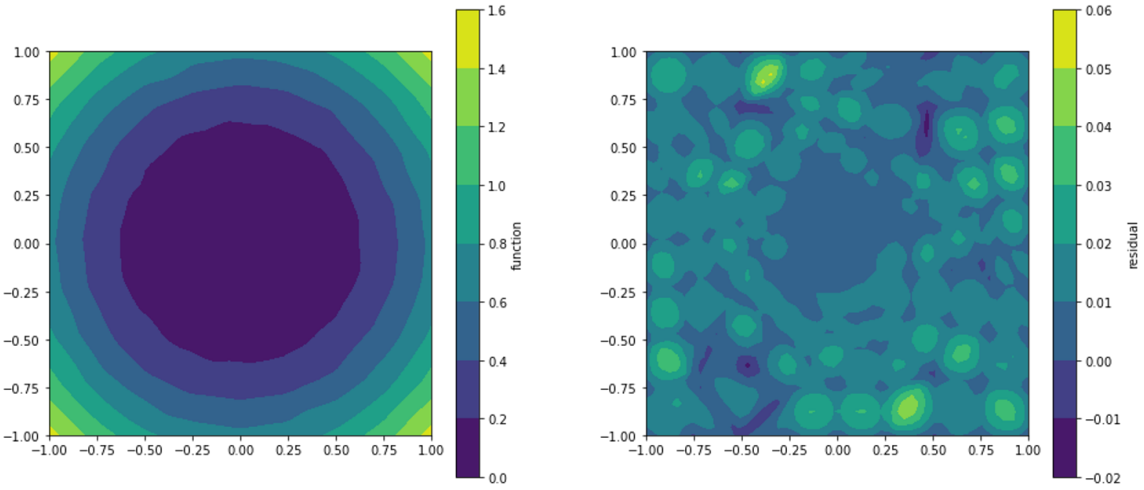


FIG. 2: Left: FEM solution of the heat-propagation problem on a squared plate with a hole for a coarse mesh. Right: Deviation of the FEM solution from the exact one.

run	training pts domain	training pts boundary	pts test	intern layers	epochs	optimizer	lern. rate
6-10	250	90	340	10 × 4	1000	adam + L-BFGS-B	1.00e-02

run	test err.	sol. deviat.	compil. time	train. time	FLOPs
6	1.42e-04	1.99e-03	1.5 s	2.3 s	4.45e+09
7	6.79e-04	1.75e-02	2.1 s	2.9 s	4.47e+09
8	1.32e-04	1.69e-03	1.3 s	2.2 s	4.38e+09
9	2.42e-04	1.60e-03	1.9 s	2.4 s	4.49e+09
10	5.48e-04	9.07e-04	1.4 s	2.7 s	4.45e+09

TABLE IV: Hyperparameters of the second simulation block (top panel) and corresponding results (bottom panel).

multiplies it for the estimated number of iterations. This procedure does not take into account for instance any internal optimization (like vectorization) and the estimate for the model, as obtained from tensorflow, currently lacks a comparison with a theoretical value [23]. It is therefore to be considered as an upper limit.

Reproducing the accuracy of the FEM solution on run 10 turned out particularly challenging with the accessible parameters: the convergence is usually fast but improving the accuracy beyond 0.01 proved to be almost impossible. We thus resorted to accessing hidden optimization parameters and used a different optimization algorithm for refinement: i.e. on top of the "adam" algorithm we performed a subsequent refinement optimization with "L-BFGS-B". (see tab. IV).

In order to reproduce a similar situation upon going from NN-runs 1-5 to 6-10, the number of training points was increased and the network was enlarged, in particular upon increasing the number of layers. Nevertheless, increasing the number of training points by a similar factor (1300) as in going from FEM runs 1 to 10 turned out to be counterproductive as it significantly affects training time without providing evident benefit in terms of resulting accuracy. It is apparent that the NN-simulations slightly

underperforms FEM run 10 in terms of accuracy on a grid of points (column "sol. deviat."). It is also worth to notice that this time the test error (column "test err.") is smaller than the true-solution deviation on the grid. This can be again traced back to the fact that these two types of errors are indeed not comparable.

The above results call for a more systematic investigation of the effect of the hyperparameters on the simulation's performance. We thus carried out a set of 5500 simulations with random values of the hyperparameters picked within pre-defined ranges in order to explore the effect of single parameters onto the resulting accuracy and computational time.

Several indicative plots of the interplay between different hyperparameters and simulation's performance are shown in fig. 3.

The main take-away messages from this analysis are the following:

- There is a strong correlation between test accuracy and deviation from the solution but there is a fraction of cases where the solution-deviation is better than what can be predicted from test accuracy.
- Without refinement, accuracy barely correlates with computation time. The correlation with refinement is weak and suggests that a significant improvement in accuracy could require an overproportionally large increase in computation time.
- Similar observations can be made for the number of training points and of epochs: these barely affect accuracy but result in increasing computational time.
- The analysis of the learning-rate dependence suggests that it has a sweet-spot at arounf 0.03 for the "adam" optimizer. Nevertheless, the subsequent refinement appears to be barely affected by this result.
- Neural network complexity (nn_cplx) was estimated as a product of network depth and width: it seems to have a fair influence on accuracy but it also significantly affects computational time. The exploration of even more complex network will certainly require an increase of the number of training points eventually which implies another computational penalty.

The hyperparameters analysis allowed to determine the optimal set of parameters in terms of accuracy with the minimal computation time. These read as follows - training pts domain: 3600, training pts boundary: 1300, intern layers: 20×4, epochs: 10000, lern. rate: 0.015, optimizer: "adam" plus refinement with "L-BFGS-B". The simulation carried out with these values returned the record accuracy (solution deviation) of 8.5×10^{-5} within a training time of about 1100 s.

The capability of the neural network was also tested by training it directly against the true solution to assess how well it is capable of reproducing it. Therein, the same parameters as in the previous runs 6-10 (the ones with the highest accuracy) have been used.

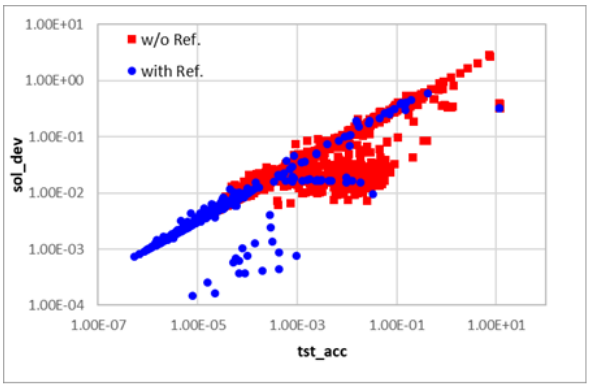
The results are summarized in tab. V. The chosen neural network can reproduce the true solution to an accuracy that is comparable but worse than that obtained upon solving the PDE, which suggests there is barely any signficant potential for improvement in the PDE-based training. This discrepancy can be possibly explained from the fact that the DeepXDE package is indeed optimized to solve differential equation and not to fit a known function.

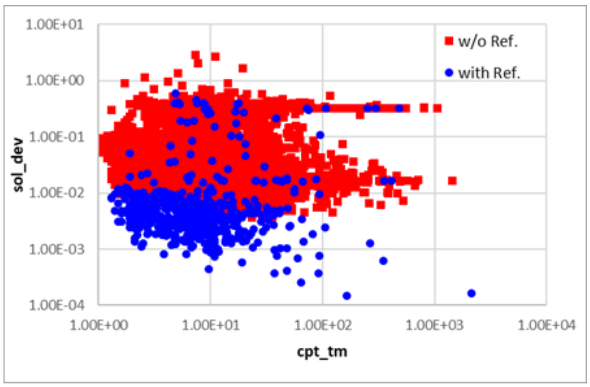
The plots in fig. 4 show the trained neural network from run 7 (the one with the best trade-off between accuracy and training time) as well as its deviation from the true solution both as a 2D-colour-map and as a 3D-surface. It is apparent that the neural network correctly reproduces the expected radial symmetry of the exact solution and the overall deviation from it has no evident pattern hinting towards the absence of systematic errors. It is important to notice how quickly the result deviates from the analytical solution in the region $r < a$, which is outside the definition domain of the problem. This indicates that extrapolations of the results outside the domain without first extending the training to the regions of interest can be unreliable.

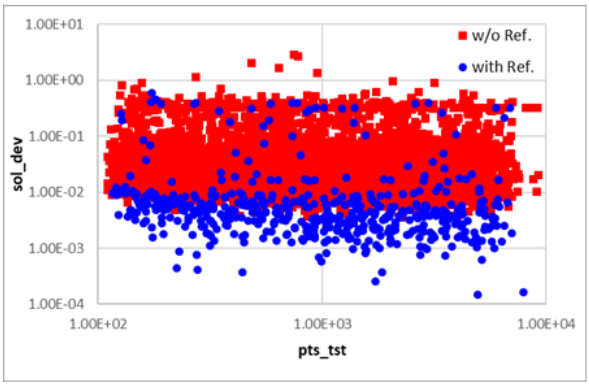
In the following we further cross-check the training on the random points and on the FEM elements. To this purpose, we extract the centroids of the FEM elements and use them as testing points for the NN-model (see fig. 5). As an additional check we train the NN-model on the FEM-centroids and test the result on random-generated points. The performance of the NN trained on the mesh centroids is represented in figs. 6 and 7. Both procedures return very similar results thus suggesting that the overall performance of a trained NN-model is globally valid. This also indicates that for more practice oriented problems a hybrid approach can be envisaged where one first solves the base geometry with FEM, thus generating a mesh whose centroids can be used as training points for the NN. Upon exploring changes in the geometry and other physical parameters of the system the NN-solution can be exploited iteratively and the need for a re-mesh, as required by FEM, can be overcome.

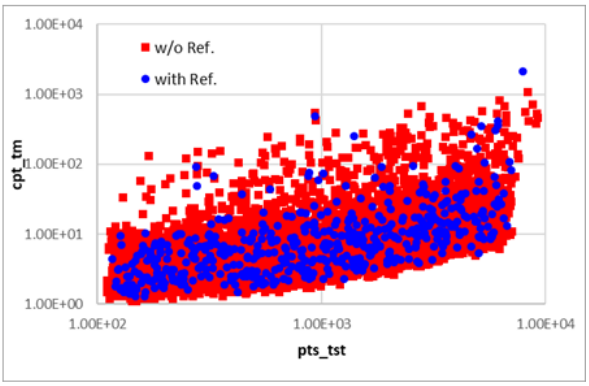
IV. CONCLUSION

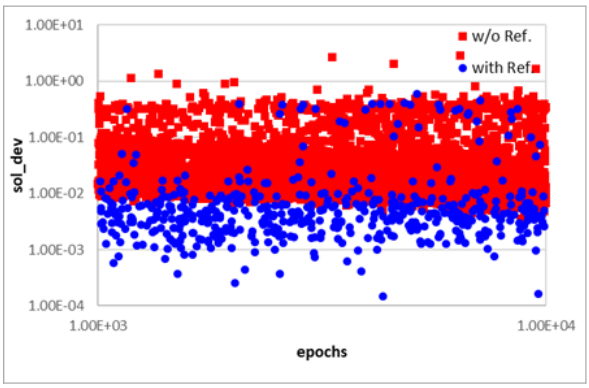
In this study we compared quantitatively the accuracy and computational performance of solving a PDE problem by means of FEM and NN for the first time by means of tackling a simple but reasonably realistic problem that admits an analytical solution. Taking the FEM calculation as a reference, the deviation from the true solution can span between 0.025 and 0.001. The former value is easily achieved with NN too without much tuning of the

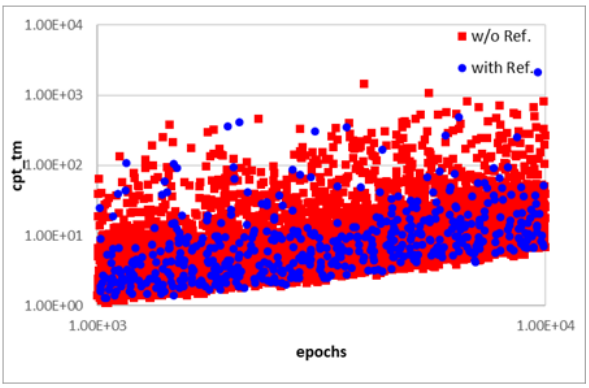

w/o Ref.
with Ref.
sol_dev
tst_acc

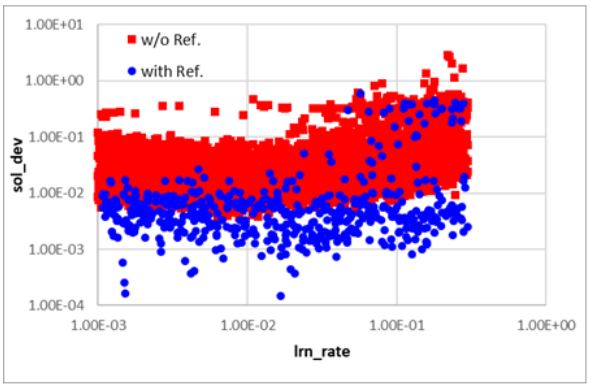

w/o Ref.
with Ref.
sol_dev
cpt_tm

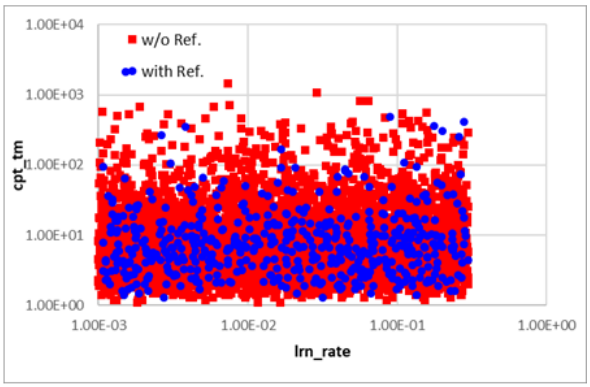

w/o Ref.
with Ref.
sol_dev
pts_tst

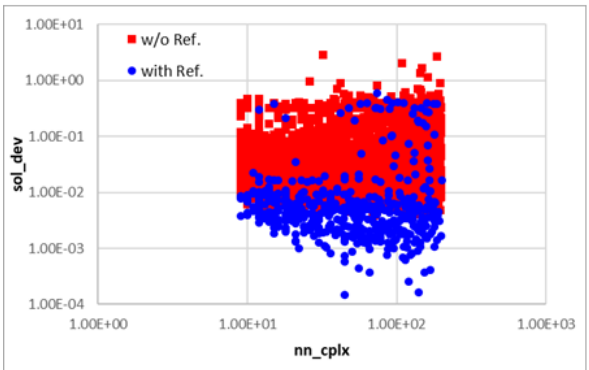

w/o Ref.
with Ref.
cpt_tm
pts_tst

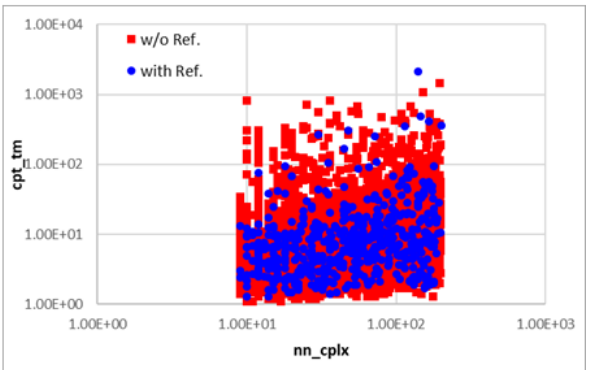

w/o Ref.
with Ref.
sol_dev
epochs


w/o Ref.
with Ref.
cpt_tm
epochs


w/o Ref.
with Ref.
sol_dev
lrn_rate


w/o Ref.
with Ref.
cpt_tm
lrn_rate


w/o Ref.
with Ref.
sol_dev
nn_cplx


w/o Ref.
with Ref.
cpt_tm
nn_cplx

run	training pts domain	training pts boundary	pts test	intern layers	epochs	optimizer	lern. rate
1-5	50	40	90	20 × 1	1000	adam	1.00e-02
6-10	250	90	340	10 × 4	1000	adam	1.00e-02

run	test err.	sol. deviat.	compil. time	train. time
1	4.16e-03	6.22e-02	0.3 s	0.6 s
2	5.22e-03	7.65e-02	0.3 s	0.6 s
3	3.19e-03	5.70e-02	0.3 s	0.6 s
4	4.15e-03	5.51e-02	0.3 s	0.6 s
5	3.98e-03	6.28e-02	0.3 s	0.6 s
6	7.67e-05	8.95e-03	0.5 s	1.9 s
7	5.40e-05	7.97e-03	0.5 s	1.9 s
8	5.05e-05	7.04e-03	0.4 s	2.5 s
9	3.68e-05	5.77e-03	0.5 s	2.9 s
10	2.40e-05	5.16e-03	0.5 s	3.2 s

TABLE V: Hyperparameters (top panel) of the direct training to the exact solution and corresponding results (bottom panel).

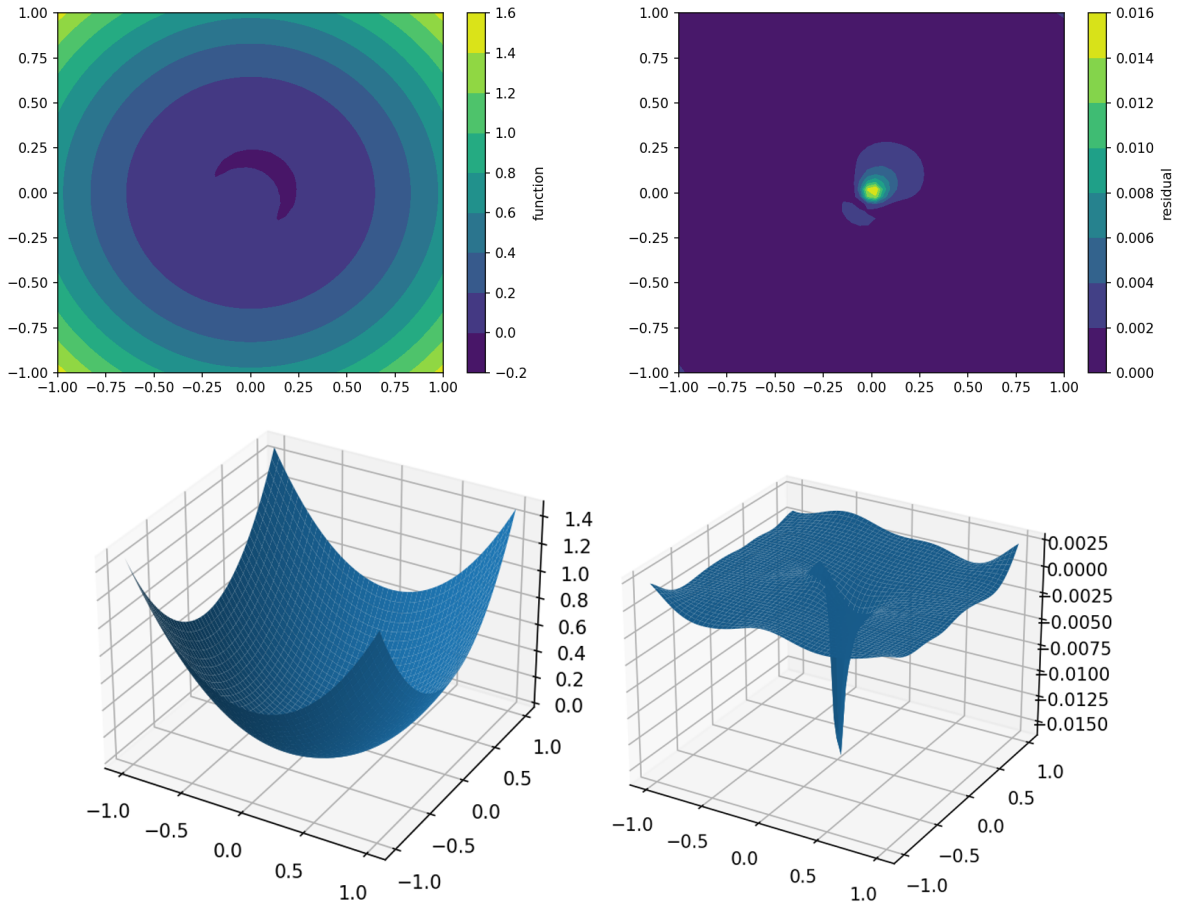


FIG. 4: NN-based direct fit of the true solution and its residual both as a colour-map and as 3D-plot. The color map of the residual represents its absolute value.

hyperparameters. The latter accuracy value resulted unachievable by means of the "adam" optimizer and only a subsequent refinement with the "L-BFGS-B" optimizer allowed to reach a similar accuracy. The reason for this limitation is currently unclear.

While it wasn't possible to accurately measure the FLOPs required by the NN simulation, thus allowing a machine-independent comparison of the computational

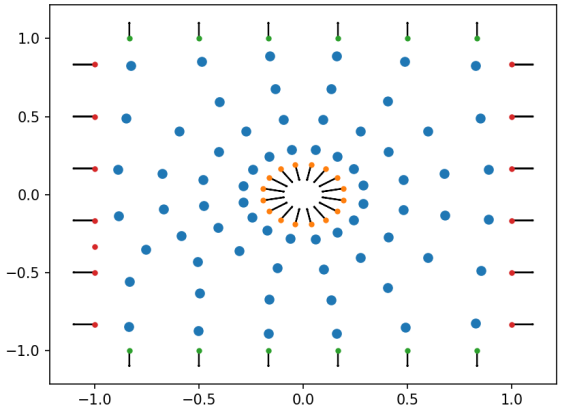


FIG. 5: Centroids of the mesh-elements used as training points for the NN. Blue points: domain; Red, green, and orange points: Boundaries. Boundaries' normal directions for von Neumann conditions are also indicated.

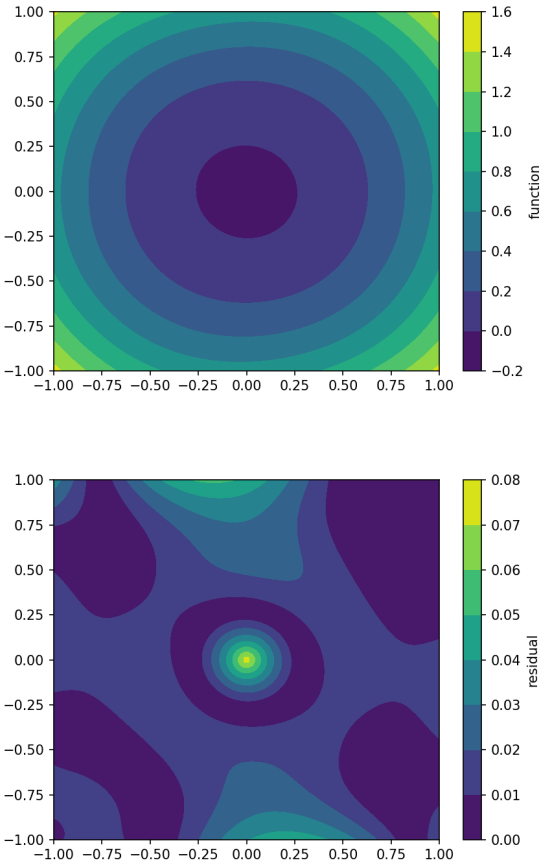


FIG. 6: PDE-NN solution obtained on the FEM-centroids and its residual (absolute value) as a colour-map.

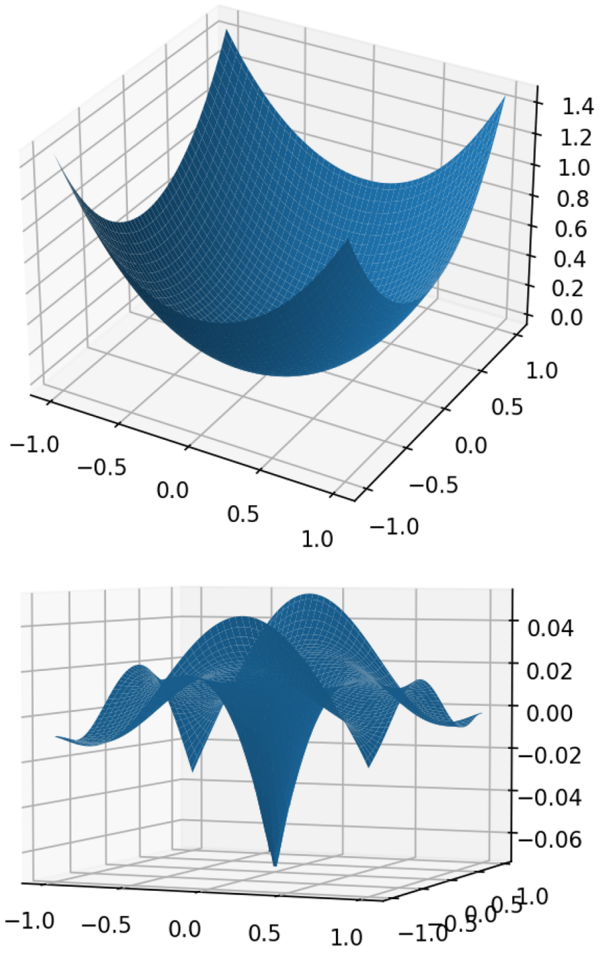


FIG. 7: PDE-NN solution obtained on the FEM-centroids and its residual as a 3D-plot.

performance, a rough estimate suggests that FEM overperforms NN significantly in this respect, as expected. It is worth to mention once more that the maturity of the available FEM-packages as compared to the PDE-NN ones can explain at least partly why FEM has a clear edge. Moreover, the potential advantages of the NN method are rather expected in subsequent repetitions of the simulation with slight changes of the underlying geometry (like in a design optimisation) than in the single run.

A detailed analysis of the effect of varying different hyperparameters shows that most of them only very weakly affect accuracy and computational time for the NN approach. Accuracies below 0.01 cannot be achieved with the "adam" optimizers and it looks as though accuracies below 10^{-5} cannot be achieved at all.

Future work aimed at definitely determining whether this approach is suitable for practical applications shall include the following steps:

- Refine the FLOPs calculation in the NN both theoretically and numerically to ensure a solid and re-

liable comparison with FEM.

- Compare the two methods on more complex calculation, e.g. upon using a much finer mesh in FEM and targeting a similar accuracy with NN.
- Optionally generate problems with analytical solutions on a more complex geometry: this is possible if one starts from an analytical function and constructs an *ad-hoc* PDE from it, including boundary conditions.
- Find possible ways to introduce a position dependent parameter (i.e. like a material property).
- Elaborate strategies for representative sampling of points from an arbitrary, complex geometry (higher point-density in smaller/sharper regions as well as at interfaces).
- Solve and compare a "real world" example with particular attention to solid-mechanics problems that represent the vast majority of FEM applications in the practice.

---

[1] R.A. Gingold; J.J. Monaghan *Mon. Not. R. Astron. Soc.* **181**, 375–89 (1977).
[2] I.E. Lagaris, A. Likas, D.I. Fotiadis *IEEE Trans. on Neural Networks* **9**, 987 (1998).
[3] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind *J. of Mach. Lear. Res.* **18**, 1-43 (2018).
[4] F. Pedregosa et al. *J. of Mach. Lear. Res.* **12**, 2825 (2011).
[5] M. Abadi et al. tensorflow.org (2015).
[6] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library" NeurIPS (2019).
[7] K. S. McFall and J. R. Mahan *IEEE Trans. on Neur. Net.* **20**, 1221 (2009).
[8] E. Samaniego et al. *Comp. Meth. in Appl. Mech. and Eng.* **362**, 112790 (2020).
[9] A. Al-Aradi, A. Correia, D. Naiff, and G. Jardim *PNAS* **115**, 8505 (2018).
[10] J. Berg, K. Nyström *Neurocomputing* **317**, 28 (2018).
[11] M.A. Nabian and H. Meidani *Probab. Eng. Mech.* **57**, 14 (2019).
[12] J. Sirignano and K. Spiliopoulos *J. of Comput. Phys.* **375**, 1339 (2018).
[13] S. Dong and Z. Li *Comp. Meth. in Appl. Mech. and Eng.* **387**, 114129 (2021).
[14] A. Dedner, R. Klöfkorn *Commun. Appl. Math. Comput.* (2021) https://doi.org/10.1007/s42967-021-00134-5
[15] Y. Shin, J. Darbon, and G. Em Karniadakis *Comm. in Comput. Phys.* **28**, 2042 (2020).
[16] D. W. Abueidda, Q. Lu, and S. Koric *Intern. J. for Num. Met. in Eng.* doi:10.1002/nme.6828 (2021).
[17] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis *SIAM Rev.* **63**, 208 (2021).
[18] R. Khudorozhkov, S. Tsimfer, A. Koryagin https://github.com/analysiscenter/pydens (2019).
[19] F. Chen et al. *J. of Open Source Soft.* **5**, 1931 (2020).
[20] W. Peng et al. arXiv:2107.04320 (2021).
[21] Dassault Systèmes Simulia Corp. "ABAQUS/Standard User's Manual, Version 2021" (2009).
[22] J. Fish and T. Belytschko, "A First Course in Finite Elements" John Wiley & Sons, Ltd (2007).
[23] An empirical dependence of the model-FLOPs for a given layer structure has been established by estimating the FLOPs width tensorflow for different network with and depth.