Tensorial Implementation for Robust Variational Physics-Informed Neural Networks

Askold Vilkha[1][0000−0001−9272−9082], Carlos Uriarte[2][0000−0002−6962−6883], Paweł Maczuga[1][0000−0002−5111−6981], Tomasz Służalec[1][0000−0001−6217−4274], and Maciej Paszyński[1][0000−0001−7766−6052]

[1]AGH University of Krakow, Poland
[2]Basque Center for Applied Mathematics, Bilbao, Spain
maciej.paszynski@agh.edu.pl

Abstract. Variational Physics-Informed Neural Networks (VPINN) train the parameters of neural networks (NN) to solve partial differential equations (PDEs). They perform unsupervised training based on the physical laws described by the weak-form residuals of the PDE over an underlying discretized variational setting; thus defining a loss function in the form of a weighted sum of multiple definite integrals representing a testing scheme. However, this classical VPINN loss function is not robust. To overcome this, we employ Robust Variational Physics-Informed Neural Networks (RVPINN), which modifies the original VPINN loss into a robust counterpart that produces both lower and upper bounds of the true error. The robust loss modifies the original VPINN loss by using the inverse of the Gram matrix computed with the inner product of the energy norm. The drawback of this robust loss is the computational cost related to the need to compute several integrals of residuals, one for each test function, multiplied by the inverse of the proper Gram matrix. In this work, we show how to perform efficient generation of the loss and training of RVPINN method on GPGPU using a sequence of einsum tensor operations. As a result, we can solve our 2D model problem within 350 seconds on A100 GPGPU card from Google Colab Pro. We advocate using the RVPINN with proper tensor operations to solve PDEs efficiently and robustly. Our tensorial implementation allows for 18 times speed up in comparison to *for*-loop type implementation on the A100 GPGPU card.

Keywords: Robust Variational Physics Informed Neural Network, GPGPU, Parallelization of tensor operations

1 Introduction

Recently, there has been a growing interest in designing and training Deep Neural Networks (DNN) for solving challenging Partial Differential Equations (PDEs). The most popular methods for training the DNN solutions of PDEs are Physics Informed Neural Networks (PINN) [1–3], and Variational Physics Informed Neural Networks (VPINN) [4]. Since their introduction in 2019, they have gained exponential growth in the number of papers and citations. It is an attractive alternative for solving PDEs, in comparison with traditional solvers such as the Finite Element Method. With the introduction of modern stochastic optimizers such as ADAM [5] they easily find high-quality minimizers of the loss functions employed.

Physics-Informed Neural Network, proposed in 2019 by Prof. Karniadakis, revolutionized the way in which neural networks find solutions to boundary-value problems described by means of PDEs [1]. In the PINN method, the neural network is treated as a function approximating the solution of a PDE. After computing the necessary differential operators, the neural network and its appropriate differential operators are inserted into the PDE. The residuum of the PDE and the boundary conditions are assumed as the loss function. The learning process consists of sampling the loss function at different points by calculating the PDE residuum and the boundary conditions. PINNs have been successfully applied to solve a wide range of problems, from fluid mechanics [2, 3], in particular, Navier-Stokes equations [6], wave propagation [7, 8], phase-field modeling [9], biomechanics [10], quantum mechanics [11], electrical engineering [12], problems with point singularities [13], uncertainty qualification [14], dynamic systems [15, 16], or inverse problems [17, 18], among many others.

Prof. Karniadakis has also proposed Variational Physics Informed Neural Networks VPINN [4]. VPINN uses the idea of a variational formulation in which the PDE is averaged using the integration over a given domain with prescribed distributions, called the test functions. The relation between PINN and VPINN is similar to the relation between finite difference and finite element methods (FDM/FEM). In the first class of methods, the continuous PDE is considered in the strong form at the set of selected points. In the second class of methods, the PDE is considered in the weak form, averaged using a family of distributions called the test functions. The VPINN method has found several applications, from Poisson and advection-diffusion equations [19], non-equilibrium evolution equations [20], solid mechanics [21], fluid flow [22], and inverse problems [23, 24], among others.

In this paper, we focus on the VPINN method. We show that the loss functions employed by the VPINN method are not robust. The loss function of VPINN can significantly differ from the true error. Thus, we employ the robust loss proposed in the RVPINN method [25]. The authors in [25] show that the robust loss proposed there is a lower bound for the true error. It is also an upper bound up to some oscillatory term.

The drawback of this robust loss is the computational cost related to the need to compute several integrals of residuals, one for each test function, multiplied by the inverse of the proper Gram matrix. In this paper, we focus on model Laplace problems. As it is shown in [25], for this kind of problem, the Gram matrix has to be computed in the weighted H^1_0 inner product. In this paper, we select the trigonometric test functions defined over the entire domain. These test functions result in the diagonal Gram matrix, as well as its inverse. We show

how to perform efficient generation of the loss on GPGPU using a sequence of einsum tensor operations.

Using our parallel tensor operations designed for RVPINN, we can solve our model 2D PDEs within 350 seconds on A100 GPGPU card from Google Colab Pro. Our parallel implementation allows for 18 times speed up in comparison to *for*-loop type implementation on the A100 GPGPU card executed using Google Colab Pro.

2 Neural-network framework

To numerically approximate PDEs, we consider a DNN function with input $\mathbf{x} = (x_1, \ldots, x_d)$ and output $u_{NN}(\mathbf{x}; \theta)$, where $\theta \in \mathbb{R}^S$ represents the trainable parameters. We employ a fully-connected Feed-Forward Neural Network (NN) composed of L layers. Each layer l in the NN consists of a set of neurons. The output of layer l, with $l = 1, \ldots, L-1$, is given by:

$$\mathbf{z}^{(l)} = \sigma(\mathbf{w}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}), \tag{1}$$

where σ is a tanh activation function, $\mathbf{w}^{(l)}$, $\mathbf{b}^{(l)}$ are the weights and biases, respectively, associated with the layer l, and $\mathbf{z}^{(0)} = \mathbf{x}$ is the input to the first layer. The final layer L is innactivated:

$$u_{NN}(\theta) = \mathbf{w}^{(L)}\mathbf{z}^{(L-1)} + \mathbf{b}^{(L)}. \tag{2}$$

Using ADAM optimization algorithm [5], the NN weights and biases are learned. We denote the manifold of different realizations of the neural network functions as U_{NN}.

3 Numerical results for VPINNs

In this section we solve two model two-dimensional problems by using VPINN [4] method. The goal of this section is to show the lack of robustness of the VPINN loss. For that, we illustrate the discrepancy between the VPINN loss function and the true error. By the true error we mean the relative error in the energy-norm defined by

$$\frac{\|u_{NN} - u_{EXACT}\|_{H_0^1}}{\|u_{EXACT}\|_{H_0^1}}, \tag{3}$$

where $\|u\|_{H_0^1} = \int_\Omega \nabla u(\mathbf{x}) \cdot \nabla u(\mathbf{x})\, d\mathbf{x}$ is the norm of the underlying trial space $H_0^1 = H_0^1(\Omega)$. Besides its theoretical foundations described in [25], this energy norm gives a good estimate how the derivatives of the NN solution approximate the derivatives of the exact solution.

3.1 Laplace model problem with sin-sin solution

Given $\Omega = (0,1)^2 \subset \mathbb{R}^2$ we seek the solution of the model problem with manufactured solution

$$-\Delta u = f, \tag{4}$$

with homogeneous Dirchlet boundary conditions that we enforce on the NN in a strong way, following the ideas presented in [26]. In this subsection, we select the solution

$$u(x_1, x_2) = sin(2\pi x_1) sin(2\pi x_2), \tag{5}$$

presented in Figure 1. In order to obtain this solution, we consider the source

$$f(x_1, x_2) = -\Delta u(x_1, x_2) = -8\pi^2 sin(2\pi x_1) sin(2\pi x_2). \tag{6}$$

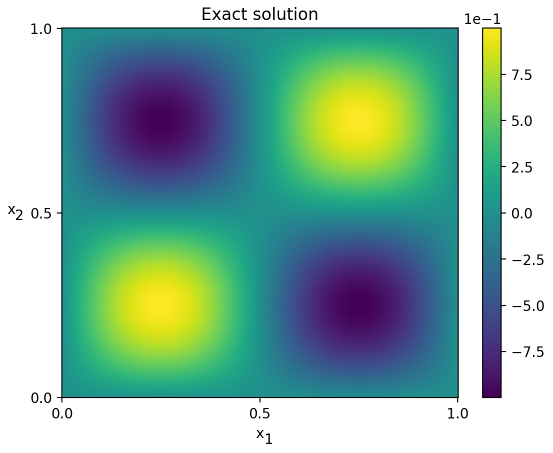

Fig. 1: Solution to the first problem.

We consider the weak form of the PDE, obtained by integration by parts with selected test functions $v \in V = H_0^1(\Omega)$. We also assume, that the solution u is approximated by the neural network $u \approx u_{NN}(\theta) \in U_{NN}$. Namely, find $u_{NN}(\theta) \in U_{NN}$ such that

$$b(u_{NN}(\theta), v) := \int_\Omega \nabla u_{NN}(\mathbf{x}) \cdot \nabla v(\mathbf{x})\, d\mathbf{x} \tag{7}$$

$$= \int_\Omega f(\mathbf{x})\; v(\mathbf{x})\, d\mathbf{x} =: l(v),\ \forall v \in V, \tag{8}$$

where $\mathbf{x}$ is an abbreviation for (x_1, x_2).

This weak formulation can equivalently be read as vanishing the following residual form:

$$r(u_{NN}(\theta), v) := b(u_{NN}(\theta), v) - l(v) = 0,\ \forall v \in V. \tag{9}$$

For the test discretization setting, we define the finite-dimensional space $V_M \subset V$

$$V_M = \mathrm{span}(\{v_m\}_{m=1}^M) \tag{10}$$

of trigonometric test functions $v_m = \sin(m_1 \pi x) \sin(m_2 \pi y)$ with $m = (m_1, m_2)$ and $1 \leq m_1 \leq M_1$ and $1 \leq m_2 \leq M_2$. Thus, $M = M_1\ M_2$.

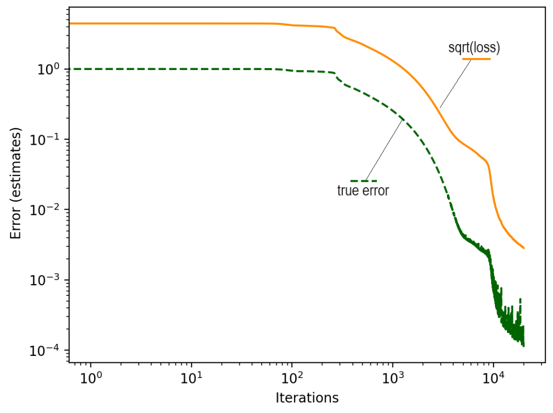

Fig. 2: First problem. VPINNs approximation using strong imposition of boundary conditions, 30×30 trigonometric test functions, 400×400 integration points for training, and 400×400 integration points for the true error. Convergence of the loss function and the convergence of the true relative error as measured in the energy norm.

In this way, the original VPINN loss function (see [4, 27]) is defined as the result of adding up all the squared residual contributions for each test basis function as follows:

$$LOSS(\theta) = \sum_{m=1}^{M} \{r(u_{NN}(\theta), v_m)\}^2 . \tag{11}$$

We perform numerical integration to approximate each residual contribution employing Monte Carlo integration, i.e.,

$$r(u_{NN}(\theta), v_m) \approx \frac{1}{K} \sum_{k=1}^{K} \nabla u_{NN}(\mathbf{x}_k) \cdot \nabla v_m(\mathbf{x}_k) - f(\mathbf{x}_k)\ s_m(\mathbf{x}_k), \tag{12}$$

where K is the total number of integration points.

For the VPINN approximation we use strong imposition of boundary conditions, 30×30 trigonometric test functions, 400×400 integration points for training, and 400×400 integration points for computing the true error for the convergence plot. Figure 2 show the convergence of the VPINN minimization. We can see from this figure that the plot of the true error $\|u_{NN} - u_{EXACT}\|$ is far from the square root of the loss (as well as from the loss itself). Here u_{NN} is the neural network solution, u_{EXACT} is the exact solution (that is usually not known in the real problems). Ideally, we would like these two plots coincide.

3.2 Laplace model problem with sin-exp solution

Following model problem (4), we consider the solution

$$u(x_1, x_2) = -exp(\pi(x_1 - 2x_2)) \sin(2\pi x_1) \sin(\pi x_2), \tag{13}$$

whose source term is

$$f(x_1, x_2) = -\pi^2 exp(\pi(x_1 - 2x_2)) sin(2\pi x_1)(4cos(\pi x_2). \tag{14}$$

This exact solution is presented in Figure 3.

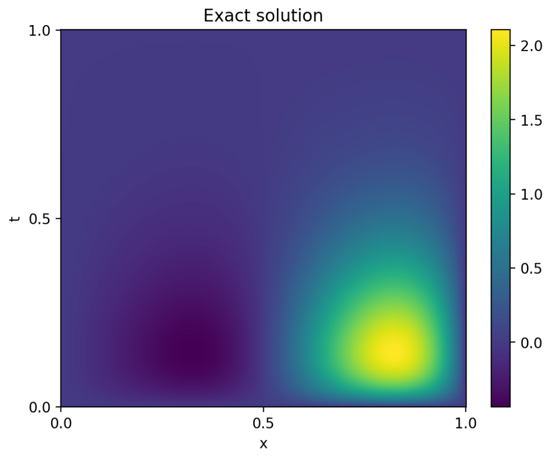

Fig. 3: Solution to the second problem.

We employ the same weak formulation as above but with different right-hand side, same test discretization setting, and same VPINN loss function. For the VPINNs approximation we use strong BCs imposition, 30×30 spectral test functions, 400×400 integration points for training, 400×400 integration points to compute the truth error for the convergence plot. The convergence of the VPINN is presented in Figure 4. We can read from this figure, that the plot of the true error $\|u_{NN} - u_{EXACT}\|$ does not coincide at all with the square root of the loss (or with the the loss itself). Here u_{NN} is the neural network solution, u_{EXACT} is the exact solution (that is usually not known in the real problems). Ideally, we would like these two plots to coincide.

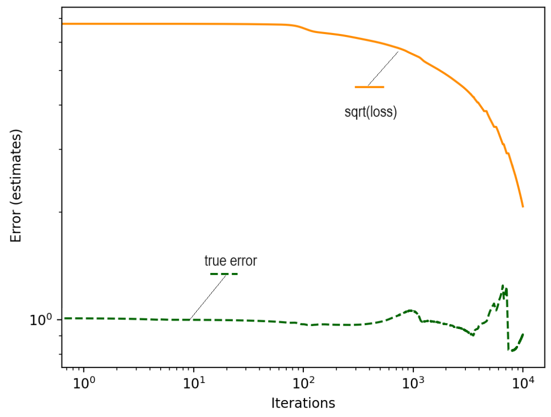

Fig. 4: Second problem. VPINNs approximation: strong BCs imposition, 30×30 spectral test functions, 400×400 integration points for training, 400×400 integration points for the true error. Convergence of the loss function and the convergence of the true relative error as measured in the energy norm.

3.3 Summary of VPINN results

The convergence of training with ADAM optimizer [5] is presented in Figures 2 and 4. We can see from these Figures that the loss functions are far from the true error computed between the approximated solution u_{NN} and the known exact solutions. Thus, the VPINN loss is not robust. Changing the number of neurons or layers, improving the quadrature as in [28], or changing the training rate, does not help to make this loss robust. Still, the VPINN loss allows us to obtain the correct solutions presented in Figures 1 and 3, but looking at the loss function convergence in Figures 2 and 4 we cannot see what is the true error of the trained solution. Thus, we do not know what is the quality of the trained solution. This is especially true if we do not know the exact solution, which is the case in practical problems.

4 Numerical results for RVPINNs

In this section, we present how to modify the originally proposed VPINN loss function in [4, 27] to obtain its robust counterpart proposed in [25]: instead of considering a single residual contribution for each selected test basis function, we have to test all the residual contributions against each other. Moreover, such testing has to be consistently weighted via the corresponding Gram matrix.

We emphasize that simply avoiding crossed multiplications of residual terms and Gram-matrix coefficients does not guarantee robustness during training, i.e., lower and upper bounds for the true error during loss minimization, as seen in the VPINN experiments of Figures 2 and 4. We refer to [25] for details.

Following a general test-space discretization V_M spanned by basis functions $\{v_m\}_{m=1}^M$, the robust version of the VPINN loss function is as follows:

$$LOSS(\theta) = \sum_{m,n=1}^{M} \{r(u_{NN}(\theta), v_m)\} \, G^{-1}_{m,n} \, \{r(u_{NN}(\theta), v_n)\} \, .$$

One might argue that original VPINNs is a simplification of RVPINNs when the Gram matrix is the identity.

In our discretization setting, it is easy to check that our trigonometric basis functions are orthogonal with respect to the inner product in $H_0^1(\Omega)$, producing a corresponding diagonal inverse of the Gram matrix as follows:

$$G^{-1}_{m_1 m_2, n_1 n_2} = \begin{cases} \frac{4}{(m_1^2 + m_2^2)\pi^2}, & \text{if } m_1 = n_1, m_2 = n_2, \\ 0, & \text{otherwise}, \end{cases} \tag{15}$$

Here, $G_{m_1 m_2, n_1 n_2} = (v_{m_1 m_2}, v_{n_1 n_2})_{H_0^1}$ denotes the Gram matrix. This reduces our additive complexity of the RVPINN loss function to

$$LOSS(\theta) = \sum_{m_1, m_2} G^{-1}_{m_1 m_2} \{r(u_{NN}(\theta), v_{m_1, m_2})\}^2 \, , \tag{16}$$

where, by abuse of notation, $G^{-1}_{m_1 m_2}$ denotes the diagonal of the inverse of the Gram matrix given by the coefficients in (15).

4.1 Laplace model problem with sin-sin solution

For the first model problem, the loss function (16) is robust. This time, the square root of the loss function is equal to the true error, as it is presented in Figure 5. This indicates that the robust loss accurately reflects the error between the neural network solution u_{NN} and the exact solution u_{EXACT}, namely $\|u_{NN} - u_{EXACT}\|$. This is true even when the exact solution is unknown. Thus, we know when to stop the training to get a good quality solution.

For the RVPINNs approximation we use strong BCs imposition, 20×20 spectral test functions, 200×200 integration points for training, 1000×1000 integration points to compute the truth error for the convergence plot. The convergence of the VPINN is presented in Figure 4.

4.2 Laplace model problem with sin-exp solution

For the second model problem, the loss function (16) is also robust. In contrary to the VPINN, illustrated in Figure 4, in the RVPINN, the square root of the loss function is a good estimate of the true error, as it is presented in Figure 6. For the VPINNs approximation we use strong BCs imposition, 20×20 spectral test functions, 200×200 integration points for training, 1000×1000 integration points to compute the truth error for the convergence plot.

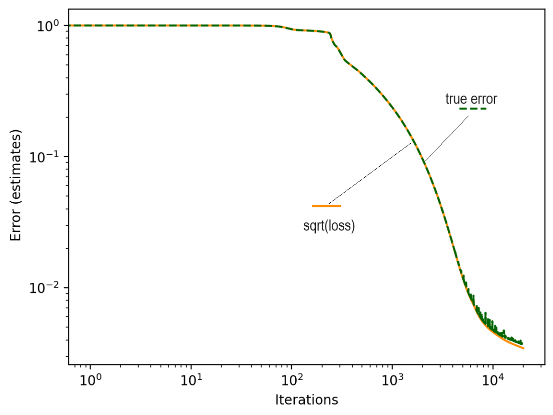

Fig. 5: First problem. RVPINNs approximation: strong BCs imposition, 20×20 spectral test functions, 200×200 integration points for training, 1000×1000 integration points for truth error. Convergence of the loss function and the convergence of the true relative error as measured in the energy norm.

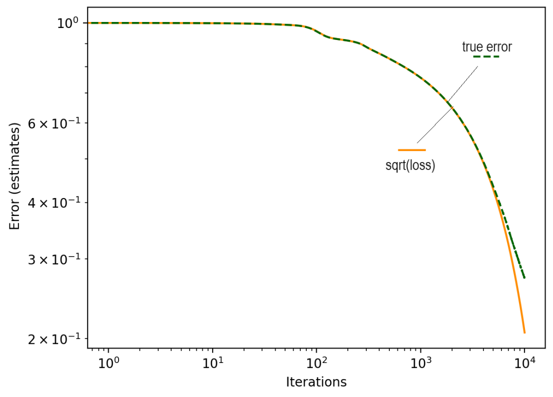

Fig. 6: Second problem. RVPINNs approximation: strong BCs imposition, 20×20 spectral test functions, 200×200 integration points for training, 1000×1000 integration points for truth error. Convergence of the loss function and the convergence of the true relative error as measured in the energy norm.

5 Tensor implementation for (R)VPINN

The RVPINN method with trigonometric test functions requires the addition of several residual terms and multiplied by the coefficients of the inverse of the Gram matrix in the loss function.

Although we can easily implement this sheme using component-by-component operations over a few number of *for*-type loops, it should be noted that such an approach is highly inefficient in interpreted programming languages like Python, which is where neural-network-based models are nowadays majorly developed. In this way, an efficient tensor workflow implementation consists of properly organizing operation functions from GPGPU-developed libraries like TensorFlow, PyTorch, or JAX. Trying to design a flowchart with operations outside these libraries typically produces disproportionately inefficient execution times. We considered PyTorch as our coding platform.

5.1 Linear algebra with Einstein summation

Our loss function consists of a combination of numerical integration, Eq. (12), and the residual summation of Eq. (16) as follows:

$$\sum_m G_m^{-1} \left\{ \frac{1}{K} \sum_{k=1}^{K} \nabla u_{NN}(\mathbf{x}_k) \cdot \nabla v_m(\mathbf{x}_k) - f(\mathbf{x}_k)\, v_m(\mathbf{x}_k) \right\}^2 .$$

where $m = (m_1, m_2)$ and $\mathbf{x}_k = (x_{1k}, x_{2k})$.

This summation expression involves the appropriate combination of tensors. These operations have a user-friendly implementation on tensor-oriented platforms that follow the Einstein summation convention.

In the following, we describe and support with graphics the implementation of our loss evaluation.

We start from the operation constructing 3D tensors. From now on, `x`, `t`, `n` and `m` replace x_1, x_2, n, and m, respectively. The `einsum` function prevents representing the indexes of each axis by more than one character during codification.

```
x_times_n = torch.einsum("xt,n->xtn",
   x.reshape(n_x, n_t), n)
t_times_m = torch.einsum("xt,m->xtm",
   t.reshape(n_x, n_t), m)
```

Now, we construct a 4D tensor with values of two-dimensional test functions, from two 3D tensors.

```
test_x = torch.sin(math.pi*x_times_n)
test_t = torch.sin(math.pi * t_times_m)
test = torch.einsum("xtn,xtm->xtnm",
   test_x, test_t)
```

Next, we construct the derivatives of the test functions along x.

```
test_x_dx = torch.pi *
```

```
   torch.einsum("n,xtn->xtn", n,
   torch.cos(torch.pi*x_times_n))
```

We also construct a 4D tensor with values of derivatives of two-dimensional test functions with respect to x, out of two 3D tensors.

```
test_dx = torch.einsum("xtn,xtm->xtnm",
   test_x_dx, test_t)
```

Next we compute derivatives of the test functions with respect to t.

```
test_t_dt = torch.pi *
   torch.einsum("m,xtm->xtm", m,
   torch.cos(torch.pi*t_times_m))
```

We also construct a 4D tensor with values of derivatives of two-dimensional test functions with respect to t, out of two 3D tensors.

```
test_dt = torch.einsum("xtn,xtm->xtnm",
   test_x, test_t_dt)
```

Finally, we construct the first part of the loss function, a 2D tensor, out of the Neural Network and the derivatives of test functions with respect to x.

```
loss1 = dx * dt * epsilon *
   torch.einsum("xt,xtnm->nm",
   dpinn_dx, test_dx)
```

We also construct in the analogous way the second part of the loss function, a 2D tensor, out of the Neural Network and the derivatives of test functions with respect to t.

```
loss2 = dx * dt * epsilon *
   torch.einsum("xt,xtnm->nm",
   dpinn_dt, test_dt)
```

The last part of the loss function is constructed out of the right-hand side 2D tensor and the 4D tensor representing the test functions.

```
loss3 = dx * dt *
   torch.einsum("xt,xtnm->nm", rhs, test)
```

We sum up all the loss contributions

```
loss = loss1 + loss2 - loss3
```

we take the second power and multiply by the Gram matrix.

```
loss = loss**2 * self.G
```

and we return the sum of all the obtained loss values.

```
return loss.sum()
```

6 Numerical experiments

6.1 Google ColabPro comparison

We perform 20,000 iterations with 100×100 integration points, 20×20 basis functions, 2 layers of the neural network with 200 neurons each. Using our tensorial implementation, we can solve the experiments conducted in Sections 3 and 4 within 350 seconds on a A100 GPGPU card from Google Colab. Execution times in a implementation using *for*-type loops takes 108 minutes. We have 18 times speed up the training process using tensor magic.

6.2 CYFRONET supercomputing center experiment

To investigate further the scalability of RVPINN we have executed our model problems on two GPUs, both NVIDIA A100-SXM4-40GB with a total Memory of 39.56 GB each, 108 multiprocessors each, with CUDA Capability: 8.0. The timing for growing number of training points is presented in Figure 7

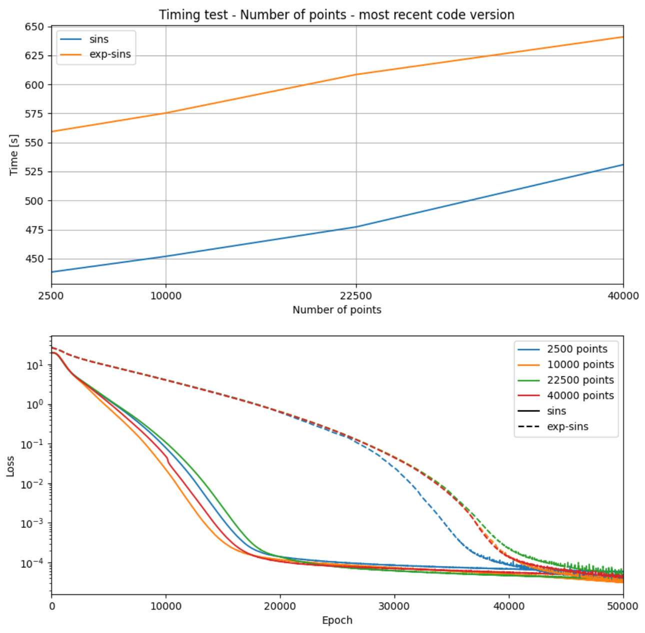

Fig. 7: RVPINNs approximation of the first and second problem, for 20×20 test functions, for growing number of integration points for training (from $50 \times 50 = 2500$, $100 \times 100 = 10000$, $150 \times 150 = 22500$, and $200 \times 200 = 400000$.

We can conclude that the 50×50 points are enough for both problems. For the first problem we need 20,000 iterations, which takes less than 450 seconds, for the second problem we need 40,000 iterations which takes less than 650 seconds.

7 Conclusions

Robust Variational Physics-Informed Neural Networks (RVPINNs) are a pivotal advancement in addressing the inherent unrobust nature of Variational Physics-Informed Neural Networks (VPINN). By recalibrating the VPINN loss function to provide a good estimation of the true error, RVPINN offers a more reliable and comprehensive estimation of solution accuracy. In addition, it is critical to implement in terms of tensor algebra in order to exploit GPGPU power during training. A typical implementation in terms of *for*-type loops is inefficient in a tensor workflow, such as in (R)VPINNs. Our tensorial implementation allows for 18 times speed up in comparison to *for*-loop type implementation on a A100 GPGPU card from Google Colab. The future work may involve extension of the method to other classical problems solved by finite element method [29, 30], and including adaptive algorithms [31–34] for the test space.

Acknowledgements

This Project has received funding from the European Union's Horizon Europe research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 101119556.

Carlos Uriarte is supported by: PID2023-146678OB-I00 funded by MICIU / AEI / 10.13039 / 501100011033 and by FEDER, EU; PID2023-146668OA-I00 funded by MICIU /AEI / 10.13039 / 501100011033 and by FEDER, EU; the BCAM Severo Ochoa accreditation of excellence CEX2021-001142-S funded by MICIU / AEI / 10.13039 / 501100011033; and the BERC 2022-2025 program funded by the Basque Government.

References

1. M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational physics 378 (2019) 686–707.
2. S. Cai, Z. Mao, Z. Wang, M. Yin, G. E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: A review, Acta Mechanica Sinica 37 (12) (2021) 1727–1738.
3. Z. Mao, A. D. Jagtap, G. E. Karniadakis, Physics-informed neural networks for high-speed flows, Computer Methods in Applied Mechanics and Engineering 360 (2020) 112789.
4. E. Kharazmi, Z. Zhang, G. E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, arXiv preprint arXiv:1912.00873 (2019).
5. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
6. J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, Journal of Fuild Mechanics 807 (2016) 155–166. doi:10.1017/jfm.2016.615.
7. M. Rasht-Behesht, C. Huber, K. Shukla, G. E. Karniadakis, Physics-informed neural networks (pinns) for wave propagation and full waveform inversions, Journal of Geophysical Research: Solid Earth 127 (5) (2022) e2021JB023120.

8. P. Maczuga, M. Paszyński, Influence of activation functions on the convergence of physics-informed neural networks for 1d wave equation, in: J. Mikyška, C. de Mulatier, M. Paszynski, V. V. Krzhizhanovskaya, J. J. Dongarra, P. M. Sloot (Eds.), Computational Science – ICCS 2023, Springer Nature Switzerland, Cham, 2023, pp. 74–88.
9. S. Goswami, C. Anitescu, S. Chakraborty, T. Rabczuk, Transfer learning enhanced physics informed neural network for phase-field modeling of fracture, Theoretical and applied fracture machanics 106 (2020). doi:10.1016/j.tafmec.2019.102447.
10. M. Alber, A. B. Tepole, W. R. Cannon, S. De, S. Dura-Bernal, K. Garikipati, G. Karniadakis, W. W. Lytton, P. Perdikaris, L. Petzold, E. Kuhl, Integrating machine learning and multiscale modeling-perspectives, challenges, and opportunities in the biologica biomedical, and behavioral sciences, NPJ Digital Medicine 2 (2019). doi:10.1038/s41746-019-0193-y.
11. H. Jin, M. Mattheakis, P. Protopapas, Physics-informed neural networks for quantum eigenvalue problems, in: 2022 International Joint Conference on Neural Networks (IJCNN), 2022, pp. 1–8. doi:10.1109/IJCNN55064.2022.9891944.
12. R. Nellikkath, S. Chatzivasileiadis, Physics-informed neural networks for minimising worst-case violations in dc optimal power flow, in: 2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), 2021, pp. 419–424. doi:10.1109/SmartGridComm51999.2021.9632308.
13. X. Huang, H. Liu, B. Shi, Z. Wang, K. Yang, Y. Li, M. Wang, H. Chu, J. Zhou, F. Yu, B. Hua, B. Dong, L. Chen, A universal pinns method for solving partial differential equations with a point source, Proceedings of the Fourteen International Joint Conference on Artificial Intelligence (IJCAI-22) (2022) 3839–3846.
14. Y. Yang, P. Perdikaris, Adversarial uncertainty quantification in physics-informed neural networks, Journal of Computational Physics 394 (2019) 136–152. doi:10.1016/j.jcp.2019.05.027.
15. F. Sun, Y. Liu2, H. Sun, Physics-informed spline learning for nonlinear dynamics discovery, Proceedings of the Thirtieth International Jint Conference on Artificial Intelligence (IJCAI-21) (2021) 2054–2061.
16. J. Kim, K. Lee, D. Lee, S. Y. Jhin, N. Park, Dpm: A novel training method for physics-informed neural networks in extrapolation, Proceedings of the AAAI Conference on Artificial Intelligence 35 (9) (2021) 8146–8154. doi:10.1609/aaai.v35i9.16992.
URL https://ojs.aaai.org/index.php/AAAI/article/view/16992
17. S. Mishra, R. Molinaro, 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) (2022) 981–1022.
18. L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, S. G. Johnson, Physics-informed neural networks with hard constraints for inverse design, SIAM Journal on Scientific Computing 43 (6) (2021) B1105–B1132. doi:10.1137/21M1397908.
19. E. Kharazmi, Z. Zhang, G. E. Karniadakis, hp-VPINNs: Variational physics-informed neural networks with domain decomposition, Computer Methods in Applied Mechanics and Engineering 374 (2021) 113547. doi:https://doi.org/10.1016/j.cma.2020.113547.
URL https://www.sciencedirect.com/science/article/pii/S0045782520307325
20. S. Huang, Z. He, B. Chem, C. Reina, Variational onsager neural networks (vonns): A thermodynamics-based variational learning strategy for non-equilibrium pdes, Journal of the mechanics and physics of solids 163 (2022). doi:10.1016/j.jmps.2022.104856.

21. C. Liu, H. A. Wu, A variational formulation of physics-informed neural network for the applications of homogeneous and heterogeneous material properties identification, International Journal of Applied Mechanics 15 (08) (2023). doi:10.1142/S1758825123500655.
22. Y. Kim, H. Kwak, J. Nam, Physics-informed neural networks for learning fluid flows with symmetry, Korean Journal of Chemical Engineering 40 (9) (2023) 2119–2127. doi:10.1007/s11814-023-1420-4.
23. C. Liu, H. Wu, cv-pinn: Efficient learning of variational physics-informed neural network with domain decomposition, Extreme Mechanics Letter 63 (2023). doi:10.1016/j.eml.2023.102051.
24. S. Badia, W. Li, A. F. Martin, Finite element interpolated neural networks for solving forward and inverse problems, Computer Methods in Applied Mechanics and Engineering 418 (A) (2024). doi:10.1016/j.cma.2023.116505.
25. S. Rojas, P. Maczuga, J. Muñoz-Matute, D. Pardo, M. Paszyński, Robust variational physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering 425 (2024) 116904. doi:https://doi.org/10.1016/j.cma.2024.116904.
URL https://www.sciencedirect.com/science/article/pii/S0045782524001609
26. L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Computer Methods in Applied Mechanics and Engineering 361 (2020) 112732. doi:https://doi.org/10.1016/j.cma.2019.112732.
URL https://www.sciencedirect.com/science/article/pii/S004578251930622X
27. E. Kharazmi, Z. Zhang, G. E. Karniadakis, hp-vpinns: Variational physics-informed neural networks with domain decomposition, Computer Methods in Applied Mechanics and Engineering 374 (2021) 113547.
28. S. Berrone, C. Canuto, M. Pintore, Variational physics informed neural networks: the role of quadratures and test functions, Journal of Scientific Computing 92 (3) (2022). doi:10.1007/s10915-022-01950-4.
29. L. Demkowicz, Computing with *hp*-adaptive finite elements, Vol. 1, Wiley, 2006.
30. L. Demkowicz, J. Kurtz, D. Pardo, M. Paszynski, W. Rachowicz, A. Zdunek, Computing with hp-ADAPTIVE FINITE ELEMENTS: Volume II Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications (1st ed.), Chapman and Hall/CRC, 2007.
31. A. Paszyńska, M. Paszyński, E. Grabska, Graph transformations for modeling hp-adaptive finite element method with triangular elements, in: M. Bubak, G. D. van Albada, J. Dongarra, P. M. A. Sloot (Eds.), Computational Science – ICCS 2008, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 604–613.
32. M. Paszyński, A. Paszyńska, Graph transformations for modeling parallel hp-adaptive finite element method, in: R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Wasniewski (Eds.), Parallel Processing and Applied Mathematics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 1313–1322.
33. M. Paszyński, R. Grzeszczuk, D. Pardo, L. Demkowicz, Deep learning driven self-adaptive hp finite element method, in: M. Paszynski, D. Kranzlmüller, V. V. Krzhizhanovskaya, J. J. Dongarra, P. M. A. Sloot (Eds.), Computational Science – ICCS 2021, Springer International Publishing, Cham, 2021, pp. 114–121.
34. A. Paszyńska, M. Paszyński, K. Jopek, M. Woźniak, D. Goik, P. Gurgul, H. AbouEisha, M. Moshkov, V. Calo, A. Lenharth, D. Nguyen, K. Pingali, Quasi-optimal elimination trees for 2d grids with singularities, Scientific Programming (1) (2015) 303024.