# A Deep Double Ritz Method (D$^2$RM) for solving Partial Differential Equations using Neural Networks

Carlos Uriarte[a,b,*], David Pardo[b,a,c], Ignacio Muga[d,a], Judit Muñoz-Matute[a,e]

[a] *Basque Center for Applied Mathematics (BCAM), Alameda Mazarredo 14, E48009 Bilbao, Spain*
[b] *Universidad del País Vasco/Euskal Herriko Unibertsitatea (UPV/EHU), Barrio Sarriena, E48940 Leioa, Spain*
[c] *Basque Foundation for Science (Ikerbasque), Plaza Euskadi 5, E48009 Bilbao, Spain*
[d] *Pontificia Universidad Católica de Valparaíso (PUCV), Avenida Brasil 2950, Valparaíso, Chile*
[e] *Oden Institute for Computational Engineering and Sciences (OICES), 201 E 24th St, 78712, Austin, TX, USA*

## Abstract

Residual minimization is a widely used technique for solving Partial Differential Equations in variational form. It minimizes the dual norm of the residual, which naturally yields a saddle-point (min–max) problem over the so-called trial and test spaces. In the context of neural networks, we can address this min–max approach by employing one network to seek the trial minimum, while another network seeks the test maximizers. However, the resulting method is numerically unstable as we approach the trial solution. To overcome this, we reformulate the residual minimization as an equivalent minimization of a Ritz functional fed by optimal test functions computed from another Ritz functional minimization. We call the resulting scheme the Deep Double Ritz Method (D$^2$RM), which combines two neural networks for approximating trial functions and optimal test functions along a nested double Ritz minimization strategy. Numerical results on different diffusion and convection problems support the robustness of our method, up to the approximation properties of the networks and the training capacity of the optimizers.
© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

*Keywords:* Partial Differential Equations; Variational formulation; Residual minimization; Ritz Method; Optimal test functions; Neural networks

## 1. Introduction

In the last decade, Neural Networks (NNs) have emerged as a powerful alternative for solving Partial Differential Equations (PDEs). For example, [1–4] employ NNs to generate optimal meshes for later solving PDEs by a Finite Element Method (FEM), [5] proposes a Deep-FEM method that mimics mesh-refinements within the NN architecture, [6] employs a NN to generate the mesh via *r*-adaptivity, and [7,8] use NNs to improve discrete weak formulations. Alternatively, there exist approaches to directly represent the PDE solutions via NNs. To mention a few: [9,10] minimize the strong form of the residual via collocation methods; [11] proposes a Deep Ritz Method (DRM) for symmetric and positive definite problems; [12] introduces a Deep Fourier Method; and [13–17] propose

(Petrov-)Galerkin frameworks in the context of trial NNs with test functions belonging to linear spaces. All these NN-based methods exhibit multiple features but also present several limitations (see, e.g., [18–21]). For example, (*hp*-)VPINNs [14,16] require a set of test functions which may be difficult to construct in order to guarantee optimal stability properties.

In the context of variational residual minimization methods [22–26], Weak Adversarial Networks (WANs) [27,28] approximate both the trial and test functions via NNs. Since the dual norm is defined as a supremum over the test space, the residual minimization reads as a min–max problem over the trial and test real Hilbert spaces, $\mathbb{U}$ and $\mathbb{V}$, respectively. Namely,

$$u^* = \arg\min_{u \in \mathbb{U}} \|Bu - l\|_{\mathbb{V}'} = \arg\min_{u \in \mathbb{U}} \max_{\|v\|_{\mathbb{V}}=1} (Bu - l)(v), \tag{1}$$

where $B : \mathbb{U} \longrightarrow \mathbb{V}'$ is the differential operator governing the PDE in variational form, $u^*$ is the exact solution, $l \in \mathbb{V}'$ is the right-hand side, and $\mathbb{V}'$ denotes the dual space of $\mathbb{V}$. This naturally leads to employing Generative Adversarial Networks (GANs) [29] by approximating $u$ and $v$ with two NNs. Unfortunately, this approach presents a severe numerical limitation: the Lipschitz continuity constant of the test maximizers with respect to the trial functions becomes arbitrarily large when approaching the exact solution. Indeed, the corresponding test maximizer is highly non-unique in the limit. This provides an inherent lack of numerical stability of the method, which is confirmed by our numerical experiments with simple model problems.

To overcome the above limitations, we reformulate the residual minimization as a minimization of a Ritz functional fed by optimal test functions [30,31]. Since optimal test functions are in general unknown, we compute them for each trial function using another Ritz method. Thus, the resulting scheme is a nested double-loop Ritz minimization method: the outer loop seeks the trial solution, while the inner loop seeks the optimal test function for each trial function. We call this the *Double Ritz Method*.

In some occasions, the trial-to-test operator that maps each trial function with the corresponding optimal test function is available. For example, when the problem is symmetric (in which case $\mathbb{U} = \mathbb{V}$), positive definite, and we select the norm induced by the bilinear form, the trial-to-test operator is the identity; or when selecting the strong variational formulation, the trial-to-test operator is the one given by the PDE operator. In these cases, the Double Ritz Method reduces to a single-loop Ritz minimization (see Section 2.4 for further details). Thus, the Double Ritz Method is a general method for solving PDEs in different variational forms, which in some particular cases simplifies into a single-loop Ritz minimization method.

Thanks to NNs, we find a simple and advantageous computational framework to approximate and connect the trial and test functions between the outer- and inner-loop minimizations in the Double Ritz Method, a task that is difficult to tackle with traditional numerical methods. In this work, we propose using one network to represent the trial functions, and another network to represent the local actions of the trial-to-test operator. Thus, the composition of both networks represents the (optimal) test functions, and we preserve the trial dependence of the test functions during the entire process. We call the resulting NN-based method the *Deep Double Ritz Method* (D²RM).

While the D²RM replicates existing residual minimization methods – and related Petrov–Galerkin methods [30–34] – in the context of NNs, we fall short to provide a detailed mathematical convergence analysis. This is because our network architectures generate manifolds instead of discrete vector spaces [35,36], which represents a departure from the traditional mathematical approach. Related to this, we encounter the usual drawbacks of lack of convexity between the trainable parameters of the network with respect to the loss functions, which is critical to make a proper diagnosis of the optimizer during training.

The remainder of this work is as follows. Section 2 derives the Double Ritz method from residual minimization at the continuous level and Section 3 introduces it in the NN framework. Section 4 provides implementation details and Section 5 describes numerical experiments. Finally, Section 6 summarizes and concludes the work.

## 2. From residual minimization to Ritz-type methods

We introduce the residual minimization approach, followed by a saddle-point formulation and an alternative Double Ritz method at the continuous level. Subsequently, we describe three particular cases for which the Double Ritz method simplifies into a single Ritz method. We follow the reasoning in [30,31] at the continuous level.

## 2.1. Residual minimization

We consider the following abstract variational formulation:

$$\left\|\begin{array}{l} \text{Find } u^* \in \mathbb{U} \text{ such that} \\ b(u^*, v) = l(v), \ \forall v \in \mathbb{V}, \end{array}\right. \tag{2}$$

where $\mathbb{U}$ and $\mathbb{V}$ are real Hilbert *trial* and *test* spaces, respectively, $b : \mathbb{U} \times \mathbb{V} \longrightarrow \mathbb{R}$ is a bilinear form, and $l : \mathbb{V} \longrightarrow \mathbb{R}$ is a continuous linear functional. Equivalently, in operator form:

$$\left\|\begin{array}{l} \text{Find } u^* \in \mathbb{U} \text{ such that} \\ Bu^* = l, \end{array}\right. \tag{3}$$

where $B : \mathbb{U} \longrightarrow \mathbb{V}'$ is the operator defined by $(Bu)(v) := b(u, v)$, $\mathbb{V}'$ denotes the topological dual of $\mathbb{V}$, and $l \in \mathbb{V}'$. The equivalent Minimum Residual formulation reads as

$$u^* = \arg\min_{u \in \mathbb{U}} \|Bu - l\|_{\mathbb{V}'}, \tag{4}$$

where $Bu - l \in \mathbb{V}'$ is the *residual* for each trial function $u \in \mathbb{U}$.

To guarantee well-posedness of problem (3), we assume

$$\{v \in \mathbb{V} : b(u, v) = 0, \forall u \in \mathbb{U}\} = \{0\}, \tag{5}$$

and that $B$ is continuous and bounded from below, i.e.,

$$\gamma \|u\|_{\mathbb{U}} \le \|Bu\|_{\mathbb{V}'} \le M \|u\|_{\mathbb{U}}, \qquad u \in \mathbb{U}, \tag{6}$$

for some positive constants $M \ge \gamma$. Then, the error in $\mathbb{U}$ is equivalent to the residual in $\mathbb{V}'$ in the following sense:

$$\frac{1}{M} \|Bu - l\|_{\mathbb{V}'} \le \|u - u^*\|_{\mathbb{U}} \le \frac{1}{\gamma} \|Bu - l\|_{\mathbb{V}'}, \qquad u \in \mathbb{U}. \tag{7}$$

However, evaluating the norm of the residual in the dual space is challenging. In what follows, we examine two alternatives to evaluate and minimize it.

## 2.2. Saddle-point problem

The norm in $\mathbb{V}'$ is defined in terms of the norm in $\mathbb{V}$ as follows:

$$\|l\|_{\mathbb{V}'} := \sup_{\|v\|_{\mathbb{V}}=1} l(v), \qquad l \in \mathbb{V}'. \tag{8}$$

Combining (4) and (8) yields:

$$u^* = \arg\min_{u \in \mathbb{U}} \max_{\|v\|_{\mathbb{V}}=1} \mathcal{J}(u, v), \qquad \mathcal{J}(u, v) := (Bu - l)(v). \tag{9}$$

In the above, the residual becomes the null operator for the exact solution, i.e., $(Bu^* - l)(v) = 0$ for all $v \in \mathbb{V}$, which implies that the test maximizer is highly non-unique in the limit. In addition, the operator that maps each trial function $u \in \mathbb{U}$ to its unitary test maximizer is not Lipschitz continuous as we approach the exact solution $u^*$ (see Appendix), leading to an unstable numerical method.

## 2.3. Double Ritz method with optimal test functions

The Riesz Representation Theorem allows us to work isometrically on the test space instead of on its dual. In particular, for the residual, we have

$$\|Bu - l\|_{\mathbb{V}'} = \|R_{\mathbb{V}}^{-1}(Bu - l)\|_{\mathbb{V}}, \tag{10}$$

where $R_{\mathbb{V}} : \mathbb{V} \ni v \longmapsto (v, \cdot)_{\mathbb{V}} \in \mathbb{V}'$ denotes the Riesz operator. This relation suggests considering the *trial-to-test* operator $T : \mathbb{U} \longrightarrow \mathbb{V}$ defined by $T := R_{\mathbb{V}}^{-1}B$, since it relates the error in $\mathbb{U}$ with the Riesz representative of the residual in $\mathbb{V}$, i.e.,

$$T(u - u^*) = R_{\mathbb{V}}^{-1}(Bu - l) \in \mathbb{V}. \tag{11}$$

The images of trial functions through $T$ are known as *optimal test functions*, and they allow us to rewrite (3) in terms of the symmetric and positive definite variational problem (cf. [31])

$$\left\|\begin{array}{l} \text{Find } u^* \in \mathbb{U} \text{ such that} \\ (Tu^*, Tu)_{\mathbb{V}} = l(Tu), \ \forall u \in \mathbb{U}, \end{array}\right. \tag{12}$$

which is equivalent to minimize the following quadratic functional:

$$u^* = \arg\min_{u \in \mathbb{U}} \mathcal{F}_T(u), \qquad \mathcal{F}_T(u) := (\mathcal{F} \circ T)(u) = \frac{1}{2}\|Tu\|_{\mathbb{V}}^2 - l(Tu). \tag{13}$$

Here, $\mathcal{F} : \mathbb{V} \longrightarrow \mathbb{R}$ is the traditional Ritz functional given by $\mathcal{F}(\cdot) = \frac{1}{2}\|\cdot\|_{\mathbb{V}}^2 - l(\cdot)$. Indeed, $\mathcal{F}_T$ acts as a generalization of the Ritz functional into more general problems (cf. Section 2.4). The relation between residual minimization (4) and the generalized Ritz minimization (13) is the following:

$$\mathcal{F}_T(u) - \mathcal{F}_T(u^*) = \frac{1}{2}\|Bu - l\|_{\mathbb{V}'}^2, \tag{14}$$

which easily deduces from the definition of $\mathcal{F}_T$.

Now, the challenge is to compute $Tu \in \mathbb{V}$ when iterating along $u \in \mathbb{U}$, since $T$ is unavailable in general. Notice that finding $Tu \in \mathbb{V}$ is equivalent to solving the symmetric and positive definite variational problem

$$\left\|\begin{array}{l} \text{Find } Tu \in \mathbb{V} \text{ such that} \\ (Tu, v)_{\mathbb{V}} = b(u, v), \ \forall v \in \mathbb{V}, \end{array}\right. \tag{15}$$

which, as before, is equivalent to the following Ritz minimization:

$$\text{Given } u \in \mathbb{U}: \quad Tu = \arg\min_{v \in \mathbb{V}} \mathcal{L}_u(v), \qquad \mathcal{L}_u(v) := \frac{1}{2}\|v\|_{\mathbb{V}}^2 - b(u, v). \tag{16}$$

However, characterizing $Tu$ by means of (16) remains impractical when iterating along $u \in \mathbb{U}$ to minimize (13), since directional derivatives of $T$ at $u \in \mathbb{U}$ are inaccessible. To overcome this, instead of iterating along $\mathbb{V}$ to seek the optimal test function for a given $u \in \mathbb{U}$, we consider a suitable set $\mathbb{M}$ of operators from $\mathbb{U}$ to $\mathbb{V}$ whose directional derivatives are accessible and the attainability of $Tu$ is guaranteed for all $u \in \mathbb{U}$. Then, we reformulate (16) as seeking a candidate $\tau_u \in \mathbb{M}$ that when acting on $u \in \mathbb{U}$ returns $Tu \in \mathbb{V}$. Namely,

$$\text{Given } u \in \mathbb{U}: \quad \tau_u = \arg\min_{\tau \in \mathbb{M}} \mathcal{L}_u(\tau(u)) \quad \text{and} \quad \tau_u(u) = Tu \in \mathbb{V}. \tag{17}$$

Notice that by performing (17), we ensure that the minimizer $\tau_u$ acts as $T$ *only* at the given $u \in \mathbb{U}$, i.e., $\tau_u(w)$ may be far from $Tw$ whenever $u \neq w$. Moreover, depending on the construction of $\mathbb{M}$, $\tau_u$ is possibly non-unique.

Hence, by performing a nested minimization of (17) within (13), we solve the problem at hand without explicitly dealing with the full operator $T$. We call this the *Double Ritz Method*. Algorithm 1 depicts its nested-loop optimization strategy that iterates separately either with elements in $\mathbb{U}$ or in $\mathbb{M}$.

---

**Algorithm 1:** Double Ritz Method

---

**1** Initialize $u \in \mathbb{U}$ and $\tau \in \mathbb{M}$;

**2** **while** not converged **do**

**3** $\quad$ Find $\tau_u \in \arg\min_{\tau \in \mathbb{M}} \mathcal{L}_u(\tau u)$;

**4** $\quad$ $u \leftarrow$ following candidate in $\mathbb{U}$ minimizing $\mathcal{F}_{\tau_u}(u)$;

**5** **return** $u$

---

### 2.4. Generalized Ritz methods

We show three scenarios where the Double Ritz Method simplifies into a single Ritz minimization, e.g., when the trial-to-test operator is available in (13).

(a) **(Traditional) Ritz Method.** If the bilinear form $b(\cdot, \cdot)$ is symmetric and positive definite, we have $\mathbb{V} = \mathbb{U}$. Selecting the inner product as the bilinear form yields $Tu = u$ for all $u \in \mathbb{U}$, i.e., $T$ is the identity operator.

(b) **Strong formulation.** Let $A : \mathcal{D}(A) \longrightarrow L^2(\Omega)$ denote a PDE operator with domain $\mathcal{D}(A)$. If we select $b(u, v) := (Au, v)_{L^2(\Omega)}$, $\mathbb{U} := \mathcal{D}(A)$ endowed with the graph norm, and $\mathbb{V} := L^2(\Omega)$, we obtain $Tu = Au$ for all $u \in \mathbb{U}$, i.e., $T$ is the PDE operator.

(c) **Ultraweak formulation.** Let $A' : \mathcal{D}(A') \longrightarrow L^2(\Omega)$ denote the adjoint operator of the PDE operator $A : \mathcal{D}(A) \longrightarrow L^2(\Omega)$, i.e., $(Au, v)_{L^2(\Omega)} = (u, A'v)_{L^2(\Omega)}$. If we select $b(u, v) := (u, A'v)_{L^2(\Omega)}$, $\mathbb{U} := L^2(\Omega)$, and $\mathbb{V} := \mathcal{D}(A')$ endowed with the graph norm, we obtain $A'Tu = u$ for all $u \in \mathbb{U}$.

Following the ideas from [34,37], we can first solve for the optimal test function of the trial solution:

$$Tu^* = \arg\min_{v \in \mathbb{V}} \mathcal{F}'(v), \qquad \mathcal{F}'(v) := \frac{1}{2}\|A'v\|^2_{L^2(\Omega)} - l(v), \tag{18}$$

and then apply $A'$ to the minimizer to recover the trial solution: $u^* = A'Tu^*$. We call *Adjoint Ritz Method* to this Ritz minimization with post-processing based on the use of the $A'$ operator.

To illustrate the above three cases, we show a simple PDE problem with different variational formulations.

**Example 1** (*Poisson's Equation*). Let $f \in L^2(0, 1)$ and consider the following 1D pure diffusion equation with homogeneous Dirichlet boundary conditions over $\Omega = (0, 1)$:

$$\begin{cases} -u'' = f, \\ u(0) = u(1) = 0. \end{cases} \tag{19}$$

We multiply the PDE by a test function and integrate over $\Omega$. Depending on the number of times we integrate by parts to derive the variational formulation, we obtain the following:

(1) **Strong formulation.** Without integration by parts. Then, $\mathbb{U} := H^2(0, 1) \cap H^1_0(0, 1)$, $\mathbb{V} := L^2(0, 1)$, $b(u, v) := \int_0^1 -u''v$, and $Tu = -u''$ for all $u \in \mathbb{U}$. This is the case (b) above.

(2) **Weak formulation.** We integrate by parts once, passing one derivative from the trial to the test function. Then, $\mathbb{U} := H^1_0(0, 1) =: \mathbb{V}$, $b(u, v) := \int_0^1 u'v'$, $(v_1, v_2)_{\mathbb{V}} := b(v_1, v_2)$, and $Tu = u$ for all $u \in \mathbb{U}$. This is the case (a) above.

(3) **Ultraweak formulation.** We integrate by parts twice, passing the two derivatives of the trial to the test function. Then, $\mathbb{U} := L^2(0, 1)$, $\mathbb{V} := H^2(0, 1) \cap H^1_0(0, 1)$, $b(u, v) := \int_0^1 -uv''$, $(v_1, v_2)_{\mathbb{V}} := (-v_1'', -v_2'')_{L^2(\Omega)}$, and $-(Tu)'' = u$ for all $u \in \mathbb{U}$. This is the case (c) above.

All three variational formulations are valid, although they exhibit different convergence behaviors that go beyond the scope of this work.

## 3. Approximation with neural networks

In practice, instead of seeking along $\mathbb{U}$, $\mathbb{V}$, and $\mathbb{M}$, we seek along corresponding computationally accessible subsets. This section elaborates on employing NNs for the computability of the proposed methods.

### 3.1. Discretization

Let $u_{\mathrm{NN}} : \Omega \longrightarrow \mathbb{R}$ denote a NN with corresponding set of learnable parameters $\theta_u$, where $\Omega$ denotes the spatial domain related to a boundary value problem. We use this NN to approximate trial functions. Thus,

$$\mathbb{U}_{\mathrm{NN}} := \{u_{\mathrm{NN}}(\cdot\,; \theta_u) : \Omega \longrightarrow \mathbb{R}\}_{\theta_u \in \Theta_u} \tag{20}$$

is the set of all possible configurations of learnable parameters (a.k.a *realizations*) for $u_{\mathrm{NN}}$, where $\Theta_u$ denotes the domain of $\theta_u$. Similarly, $v_{\mathrm{NN}} : \Omega \longrightarrow \mathbb{R}$ and $\tau_{\mathrm{NN}} : \mathbb{U}_{\mathrm{NN}} \longrightarrow \mathbb{V}_{\mathrm{NN}}$ will denote NNs with corresponding sets of learnable parameters $\theta_v \in \Theta_v$ and $\theta_T \in \Theta_T$, and sets of realizations $\mathbb{V}_{\mathrm{NN}}$ and $\mathbb{M}_{\mathrm{NN}}$, respectively. In the end, the architectures of the NNs determine the approximation capacity of our methods, but its study is beyond the scope of this work (see, e.g., [38–40]).

We select fully-connected architectures for the NNs, namely,

$$u_{\mathrm{NN}}(x_1) = L_{k+1} \circ L_k \circ \cdots \circ L_2 \circ L_1(x_1), \qquad k \geq 1, \tag{21a}$$

$$x_i = L_i(x_{i-1}) = \varphi(W_i x_{i-1} + b_i), \qquad 2 \le i \le k, \tag{21b}$$

$$L_{k+1}(x_k) = W_{k+1} x_k, \tag{21c}$$

where $x_1$ is the input vector of the network, $W_i x_{i-1} + b_i$ is an affine transformation, $\varphi$ is an activation function, $W_{k+1} x_k$ is a linear combination of the components of vector $x_k$ with coefficients in $W_{k+1}$, and $k$ is the depth of the NN. In particular, the set of learnable parameters consists of

$$\Theta_u := \{W_i \in \mathbb{R}^{n_i \times n_{i-1}}, b_i \in \mathbb{R}^{n_i \times 1} : 1 \le i \le k\} \cup \{W_{k+1} \in \mathbb{R}^{1 \times n_k}\}, \tag{22}$$

where $n_{i-1}$ and $n_i$ are the input and output dimensions of layer $L_i$.

To ensure the containment of the set of realizations within the trial space (i.e., $\mathbb{U}_{\mathrm{NN}} \subset \mathbb{U}$), we select convenient activation functions (e.g., $\varphi = \tanh$) and strongly impose Dirichlet boundary conditions whenever is needed. For the latter, we redefine the above free-boundary NN architecture as

$$u_{\mathrm{NN}}(x) \leftarrow u_D(x) + D(x) \cdot u_{\mathrm{NN}}(x), \tag{23}$$

where $D(x)$ is an auxiliary continuous function such that $D|_{\Gamma_D} = 0$ and $D|_{\overline{\Omega} \setminus \Gamma_D} \ne 0$, with $\Gamma_D$ denoting the Dirichlet boundary, and $u_D$ is a lift of $u|_{\Gamma_D}$. We consider similar architecture settings for $v_{\mathrm{NN}}$ (see Fig. 2) and $\tau_{\mathrm{NN}}$ (see Fig. 4).

Table 1 shows the methods introduced at Sections 2.2–2.4 in the framework of NNs.

**Table 1**
The min–max approach, the Generalized Ritz Method, and the Double Ritz Method in the context of NNs: *Weak Adversarial Networks (WANs), the Generalized Deep Ritz Method (GDRM), and the Deep Double Ritz Method ($D^2RM$).*

| Method | Functional(s) | Optimization |
|---|---|---|
| Weak Adversarial Networks | $\mathcal{J}(u_{\mathrm{NN}}, v_{\mathrm{NN}}) = (B u_{\mathrm{NN}} - l)\left(\dfrac{v_{\mathrm{NN}}}{\|v_{\mathrm{NN}}\|_V}\right)$ | $u_{\mathrm{NN}}^* = \arg\min\limits_{u_{\mathrm{NN}} \in \mathbb{U}_{\mathrm{NN}}} \max\limits_{v_{\mathrm{NN}} \in \mathbb{V}_{\mathrm{NN}}} \mathcal{J}(u_{\mathrm{NN}}, v_{\mathrm{NN}})$ |
| Generalized Deep Ritz Method | $\mathcal{F}_T(u_{\mathrm{NN}}) = \frac{1}{2}\|T u_{\mathrm{NN}}\|_V^2 - l(T u_{\mathrm{NN}})$ | $u_{\mathrm{NN}}^* = \arg\min\limits_{u_{\mathrm{NN}} \in \mathbb{U}_{\mathrm{NN}}} \mathcal{F}_T(u_{\mathrm{NN}})$ |
| Deep Double Ritz Method | $\mathcal{F}_{\tau_{\mathrm{NN}}}(u_{\mathrm{NN}}) = \frac{1}{2}\|\tau_{\mathrm{NN}}(u_{\mathrm{NN}})\|_V^2 - l(\tau_{\mathrm{NN}}(u_{\mathrm{NN}}))$ <br> $\mathcal{L}_{u_{\mathrm{NN}}}(\tau_{\mathrm{NN}}) = \frac{1}{2}\|\tau_{\mathrm{NN}}(u_{\mathrm{NN}})\|_V^2 - (B u_{\mathrm{NN}})(\tau_{\mathrm{NN}}(u_{\mathrm{NN}}))$ | $u_{\mathrm{NN}}^* = \arg\min\limits_{u_{\mathrm{NN}} \in \mathbb{U}_{\mathrm{NN}}} \mathcal{F}_{\tau_{u_{\mathrm{NN}}}}(u_{\mathrm{NN}})$ <br> $\tau_{u_{\mathrm{NN}}} = \arg\min\limits_{\tau_{\mathrm{NN}} \in \mathbb{M}_{\mathrm{NN}}} \mathcal{L}_{u_{\mathrm{NN}}}(\tau_{\mathrm{NN}}(u_{\mathrm{NN}}))$ |

## 3.2. Differentiation and integration

Forms $b(\cdot, \cdot)$, $l(\cdot)$, and $\|\cdot\|_{\mathbb{V}}$ consist of definite integrals of combinations of NNs and/or their derivatives. We perform *automatic-differentiation* to evaluate the derivatives of the NNs [41,42], and approximate corresponding integrals via a quadrature rule:

$$\int_\Omega I(x)dx \approx \sum_{i=1}^N \omega_i \cdot I(x_i). \tag{24}$$

Here, $I$ is the integrand and $\omega_i$ is the integration weight related to the integration node $x_i \in \Omega$. Thus, the quality of the approximation depends on the sample and the weight selection.

In Monte Carlo integration, the weights act as uniform averages of the sum of evaluations of the integrand ($\omega_i = \mathrm{Vol}(\Omega)/N$) [43,44]. It allows to select a new random sample of nodes for each integration during the training and, therefore, circumvents the well-known overfitting problem [45]. Unfortunately, Monte Carlo integration requires an immense sample size to ensure admissible integration errors, which impacts the computational memory resources and execution speed during training. In addition, Monte Carlo integration has difficulties when dealing with singular solutions that we consider in this work.

To maintain the advantages of Monte Carlo integration, reduce the runtime and sample size (from thousands to hundreds of points), and control the integration error around singularities, we consider a *composite intermediate-point quadrature rule* (see Algorithm 2 for the domain $\Omega = (0, 1)$) that generates random integration points following a beta $\beta(a, b)$ probability distribution [46,47]. We tune the hyperparameters $a$ and $b$ conveniently according to our problem specifications.

For $\Omega = (0, 1) \times (0, 1)$, we perform lines 1 and 2 of Algorithm 2 on each axis, and adapt lines 3–6 to a cartesian-product structure.

---

**Algorithm 2:** Randomized composite intermediate-point quadrature rule to approximate $\int_0^1 I(x)dx$

---

**1** Generate $x_i \in (0, 1)$ for $1 \le i \le N$;

**2** Sort $\{x_i : 1 \le i \le N\}$ so that $x_{i-1} < x_i$ for all $1 \le i \le N$;

**3** Evaluate $I$ in $\{x_i : 1 \le i \le N\}$;

**4** Define $m_0 := 0$, $m_i := (x_{i-1} + x_i)/2$ for $1 \le i \le N-1$, and $m_N := 1$;

**5** Define $\omega_i := m_i - m_{i-1}$ for $1 \le i \le N$;

**6 return** $\sum_{i=1}^N \omega_i \cdot I(x_i)$;

---

### 3.3. Loss functions and gradient-descent/ascent

We define the loss functions as sampling-dependent functionals that represent the quadrature rules approximating the analytic integrals (recall Table 1):

$$\mathcal{J}(u_{\text{NN}}, v_{\text{NN}}) \approx \widehat{\mathcal{J}}(\{x_i\}_i; \theta_u, \theta_v), \tag{25a}$$

$$\mathcal{F}_T(u_{\text{NN}}) \approx \widehat{\mathcal{F}}_T(\{x_i\}_i; \theta_u), \tag{25b}$$

$$\mathcal{F}_{\tau_{\text{NN}}}(u_{\text{NN}}) \approx \widehat{\mathcal{F}}_{\tau_{\text{NN}}}(\{x_i\}_i; \theta_u), \tag{25c}$$

$$\mathcal{L}_{u_{\text{NN}}}(\tau_{\text{NN}}(u_{\text{NN}})) \approx \widehat{\mathcal{L}}_{u_{\text{NN}}}(\{x_i\}_i; \theta_\tau). \tag{25d}$$

The right-hand sides of Eqs. (25a)–(25d) represent the loss functions, while the left-hand sides represent the corresponding analytic integrals.

Now, we detail three different optimization strategies for training the NNs:

- **Weak Adversarial Networks (WANs).** We use the Stochastic Gradient-Descent/Ascent (SGD/A) optimizer [48] to readjust the learnable parameters as follows:

$$\theta_u \leftarrow \theta_u - \eta_u \frac{\partial \widehat{\mathcal{J}}}{\partial \theta_u}(\{x_i\}; \theta_u, \theta_v), \tag{26a}$$

$$\theta_v \leftarrow \theta_v + \eta_v \frac{\partial \widehat{\mathcal{J}}}{\partial \theta_v}(\{x_i\}; \theta_u, \theta_v), \tag{26b}$$

  where $\eta_u, \eta_v > 0$ denote learning-rate coefficients. We perform multiple iterations on the ascent for each iteration on the descent (see Algorithm 3). We call *outer* and *inner loops* to the minimization and maximization processes, respectively, because of the nested optimization structure.

---

**Algorithm 3:** WANs training

---

**1** Initialize $\theta_u \in \Theta_u$ and $\theta_v \in \Theta_v$;

   /* Outer loop                                                                   */

**2 while** not converged **do**

**3**     Select random sampling $\{x_i\}_i \subset \Omega$;

**4**     $\theta_u \leftarrow \theta_u - \eta_u \frac{\partial \widehat{\mathcal{J}}}{\partial \theta_u}(\{x_i\}_i; \theta_u, \theta_v)$;

       /* Inner loop                                                                   */

**5**     **while** not converged **do**

**6**         Select random sampling $\{x_i\}_i \subset \Omega$;

**7**         $\theta_v \leftarrow \theta_v + \eta_v \frac{\partial \widehat{\mathcal{J}}}{\partial \theta_v}(\{x_i\}_i; \theta_u, \theta_v)$;

**8 return** $\theta_u$

---

- **Deep Double Ritz Method (D$^2$RM).** The training is similar as in Algorithm 3, but modifying the loss when jumping from the outer to the inner loop, and performing only gradient-descents for minimizations (see Algorithm 4).

---

**Algorithm 4:** D$^2$RM training

---

1   Initialize $\theta_u \in \Theta_u$ and $\theta_\tau \in \Theta_\tau$;

   /* Outer loop                                                                 */

2   **while** not converged **do**

3      Select random sampling $\{x_i\}_i \subset \Omega$;

4      $\theta_u \leftarrow \theta_u - \eta_u \frac{\partial \widehat{\mathcal{F}}_{\tau_{\mathrm{NN}}}}{\partial \theta_u}(\{x_i\}_i; \theta_u, \theta_\tau)$;

     /* Inner loop                                                           */

5      **while** not converged **do**

6         Select random sampling $\{x_i\}_i \subset \Omega$;

7         $\theta_\tau \leftarrow \theta_\tau - \eta_\tau \frac{\partial \widehat{\mathcal{L}}_{u_{\mathrm{NN}}}}{\partial \theta_\tau}(\{x_i\}_i; \theta_u, \theta_\tau)$;

8   **return** $\theta_u$

---

- **Generalized Deep Ritz Method (GDRM).** The optimization consists of a single-loop minimization (see Algorithm 5) since the trial-to-test operator $T$ is explicitly available. In particular, when the bilinear form is symmetric and positive definite – recall Section 2.4, item (a) –, this method is the well-known Deep Ritz Method (DRM) [11].

---

**Algorithm 5:** GDRM training

---

1   Initialize $\theta_u \in \Theta_u$;

2   **while** not converged **do**

3      Select random sampling $\{x_i\}_i \subset \Omega$;

4      $\theta_u \leftarrow \theta_u - \eta_u \frac{\partial \widehat{\mathcal{F}}_T}{\partial \theta_u}(\{x_i\}_i; \theta_u)$;

5   **return** $\theta_u$

---

- **Adjoint Deep Ritz Method (DRM)$'$.** This method consists of a single-loop minimization (as in Algorithm 5) with post-processing, only valid in ultraweak formulations. Recall Section 2.4, item (c).

### 3.4. Optimization

We select the Adam optimizer [49] to carry out our experiments. In the case of nested optimizations, we select fully independent Adam optimizers for the outer and inner loops. We establish an accumulated maximum number of iterations for both nested loops, and we fix four inner-loop iterations for each outer-loop iteration (as considered in [27,28] for WANs). In the D$^2$RM, we assume that a slight modification in $\theta_u$ translates into a slight variation in $u_{\mathrm{NN}}$. Then, the variation in $Tu_{\mathrm{NN}}$ is small because $T$ is continuous and, therefore, we expect that $\tau_{\mathrm{NN}}(u_{\mathrm{NN}})$ can provide a good approximation of $Tu_{\mathrm{NN}}$ after a small number of iterations in $\theta_\tau$.

## 4. Implementation

We use the Tensorflow 2 library (TF2) [50] within Python to implement our neural network architectures and loss functions, and to manage the random creation and flow of data. Specifically, we accommodate all of our implementations to Keras (`tf.keras`), a sublibrary of TF2 specially designed for executions in *graph mode* that exploits high-performance computing [51] (e.g., parallelized computing within GPUs [52,53]).
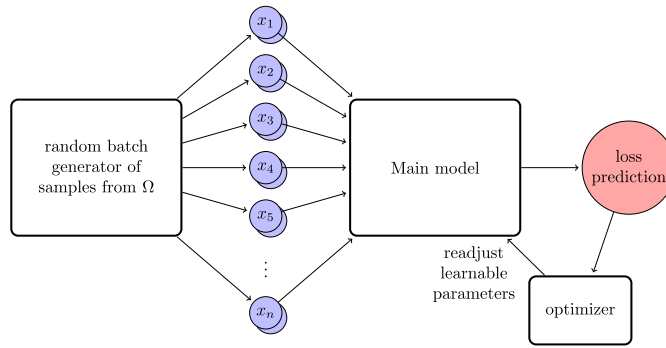
**Fig. 1.** General flux sketch for the proposed methods at each training iteration.

### 4.1. Samples generation, input batch flow, encapsulation of models, and optimization

Our inputs to networks are samples over the domain $\Omega$. After feeding our networks with the batch of inputs, we combine the batch of outputs in a single loss prediction. We encapsulate the networks and losses inside a main model whose input and output are the batch of samples and the loss prediction, respectively. From the loss prediction, we optimize the learnable parameters of the networks (i.e, we fit the learnable parameters to the data) with the Keras built-in Adam optimizer (`tf.keras.optimizers.Adam`). Fig. 1 illustrates the described process at each training iteration.

For the maximization involved in WANs, we reverse the sign of the gradients so that when feeding the (default) gradient-descent-based optimizer, it instead performs a gradient-ascent.

### 4.2. Design of models by methods

We implement networks, losses, and operators as models and layers in TF2 from the redefinitions of the corresponding Keras base classes (`tf.keras.Model` and `tf.keras.Layer`). In WANs, we implement $u_{NN}$ and $v_{NN}$ as two independent models that are combined via a common non-trainable layer for the loss (see Fig. 2). In the DRM, we implement $u_{NN}$ as a model that subsequently connects with two non-trainable layers for the trial-to-test operator and the loss, respectively (see Fig. 3). In the D$^2$RM, we implement $u_{NN}$ and $\tau_{NN}$ as two sequential models whose output feeds into two separate losses (see Fig. 4). The loss functions are implemented as latent outputs of the main model, which together with a TF2-suitable boolean variable, activate and deactivate alternatively. Despite the significant compilation time that the two-branch model (for the D$^2$RM) takes compared with one-branch models (for WANs and the DRM),[1] its fitting execution in graph mode is as fast as that of one-branch models.

### 4.3. Graph-mode execution dynamics

We employ *callbacks* to avoid interrupting the graph execution mode carried out by the Keras fitting instruction (`.fit`). Callbacks act during fitting and enable accessing certain elements of (main) models and modifying them. We utilize callbacks for loss monitoring (for WANs, the DRM, and the D$^2$RM), to activate or deactivate the trainability of the networks and switch between optimizers (for WANs and the D$^2$RM), or to interchange losses (for the D$^2$RM) during training when iterating either over the outer or the inner loop.

## 5. Numerical results

We show numerical experiments to compare the methods introduced above. Section 5.1 considers a simple model problem and initially compares WANs, the DRM, and the D$^2$RM. Section 5.2 makes a more profound comparison on a parametric problem. Sections 5.3 and 5.4 consider sources that lead to singular problems in pure diffusion and convection equations, respectively. Finally, Section 5.5 shows results on a 2D pure convection problem.

---

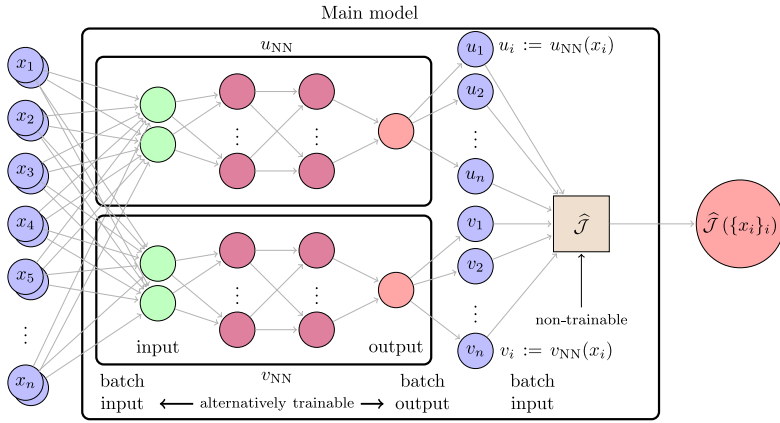[1] Around one minute for the D$^2$RM vs. a couple of seconds for WANs and the DRM.

**Fig. 2.** Main model architecture for WANs. It consists of two independent NNs, $u_{\mathrm{NN}}$ and $v_{\mathrm{NN}}$, combined via the loss $\widehat{\mathcal{J}}$.
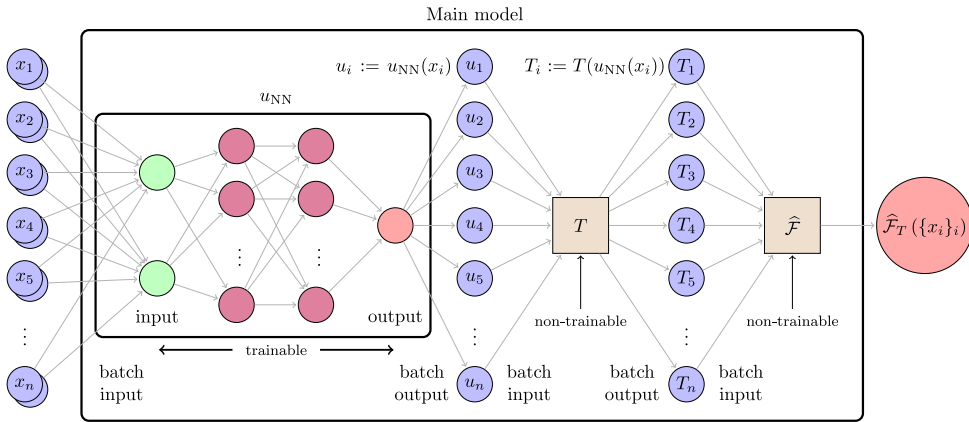


**Fig. 3.** Main model architecture for the DRM. It consists of a NN, $u_{\mathrm{NN}}$, composed with $T$ and the loss $\widehat{\mathcal{F}}$.
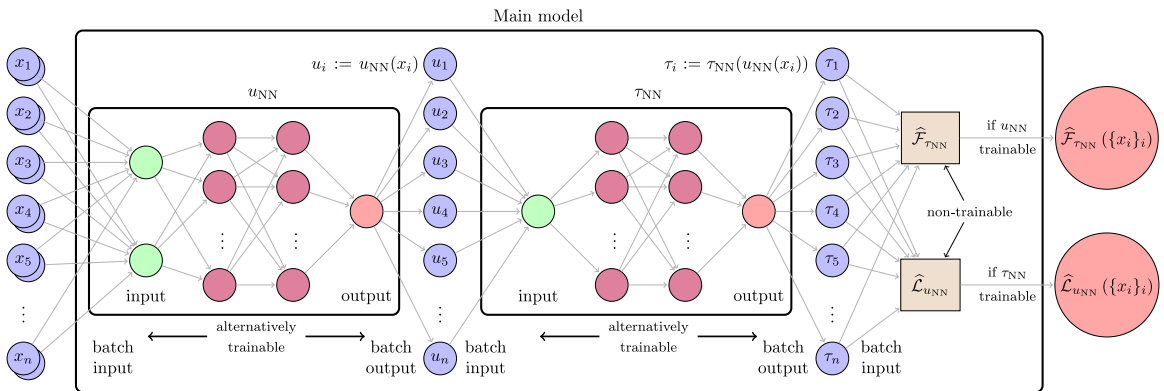


**Fig. 4.** Main model architecture for the D$^2$RM. It consists of two NNs, $u_{\mathrm{NN}}$ and $\tau_{\mathrm{NN}}$, equipped with losses $\widehat{\mathcal{F}}_{\tau_{\mathrm{NN}}}$ and $\widehat{\mathcal{L}}_{u_{\mathrm{NN}}}$.
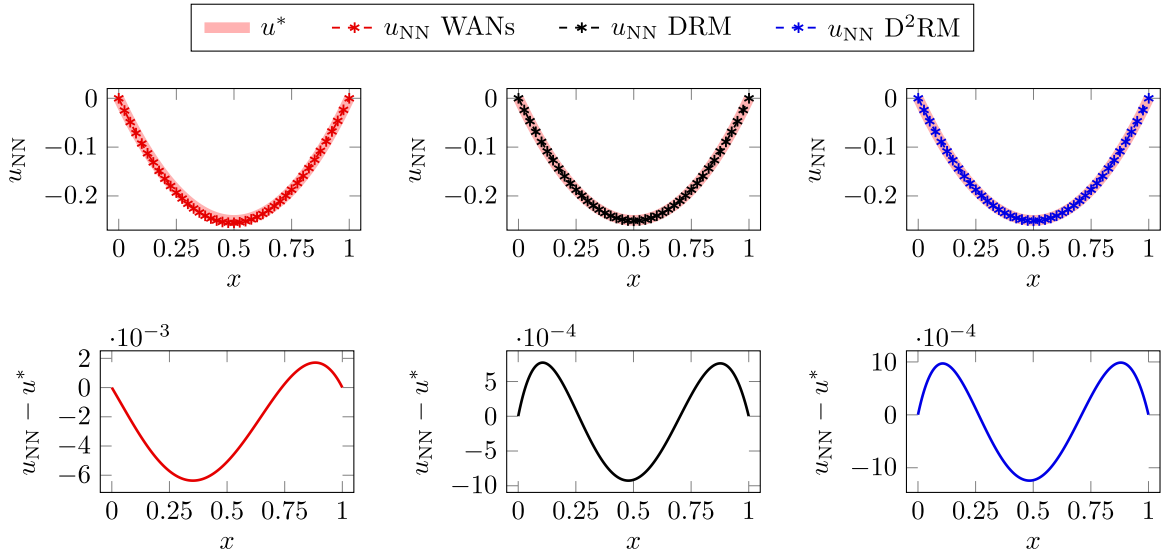
**Fig. 5.** Trial network predictions and errors in WANs, the DRM, and the D$^2$RM at the end of the training in problem (19) with analytic solution $u^* = x(x - 1)$.

### 5.1. Initial comparison of WANs, the DRM, and the D$^2$RM on a simple problem

We select model problem (19) in weak form with source $f = -2$, so the analytic solution is $u^* = x(x - 1)$. Hence, $\mathbb{U} = H_0^1(0, 1) = \mathbb{V}$ and $T$ is the identity operator. We solve this problem employing WANs, the DRM, and the D$^2$RM.

We select a two-layer fully-connected NN with 20 neurons on each layer and tanh activation functions for the architectures of $u_{\mathrm{NN}}$, $v_{\mathrm{NN}}$, and $\tau_{\mathrm{NN}}$. We perform 200 iterations for $u_{\mathrm{NN}}$ in the three methods: WANs, the DRM, and the D$^2$RM. Since in WANs and the D$^2$RM we established four iterations to approximate the test maximizers (recall Section 3.4), we end up with a total of 1000 training iterations (200 for the trial function, and 800 for the test functions). Moreover, we select batches of size 200 for the training, and a uniform distribution for the sample generation.

Fig. 5 shows the $u_{\mathrm{NN}}$ network predictions and errors of the three methods at the end of the training. We observe that WANs produce a larger error than the DRM and the D$^2$RM.

Fig. 6 shows the loss evolution for WANs. Every five iterations, the loss decreases (iterations in $u_{\mathrm{NN}}$), and increases in the remaining ones (while freezing the approximate solution and looking for the test maximizer). From iteration 500 onwards, the loss stops improving and oscillates above the optimal value.

Fig. 7 shows the loss evolution for the DRM. Here, we have a single minimization, so the observed noisy behavior of the loss towards the end of the training is due to the optimizer. Because of the lower complexity of the training, we obtain a better convergence performance than with WANs.

Fig. 8 shows the loss evolutions for the D$^2$RM. In this method, we have a min–min optimization, where both losses are minimized alternatively. At each iteration, we evaluate both $\widehat{\mathcal{F}}_{\tau_{\mathrm{NN}}}$ and $\widehat{\mathcal{L}}_{u_{\mathrm{NN}}}$, even if we are only optimizing with respect to one of them. We superimpose both losses, each one with its own scale (the left vertical axis corresponds to $\widehat{\mathcal{F}}_{\tau_{\mathrm{NN}}}$ and the right vertical one to $\widehat{\mathcal{L}}_{u_{\mathrm{NN}}}$). Both losses exhibit a decreasing staircase shape with downward-sloping steps. Jumps occur when the optimization is performed with respect to $\widehat{\mathcal{F}}_{\tau_{\mathrm{NN}}}$, which suggests that $\widehat{\mathcal{L}}_{u_{\mathrm{NN}}}$ takes longer to converge, as it depends on the convergence of $\widehat{\mathcal{F}}_{\tau_{\mathrm{NN}}}$ (outer vs. inner loop).

The relative errors of the trial network predictions[2] at the end of the training are 3.12%, 0.99%, and 1.31% in WANs, the DRM, and the D$^2$RM, respectively.

---

[2] To approximate $\frac{\|u_{\mathrm{NN}} - u\|_{\mathbb{U}}}{\|u\|_{\mathbb{U}}} \times 100$, we perform a composite intermediate-point rule with $10^4$ integration nodes for the numerator, and analytically calculate $\|u\|_{\mathbb{U}} = \sqrt{3}/3$ for the denominator.
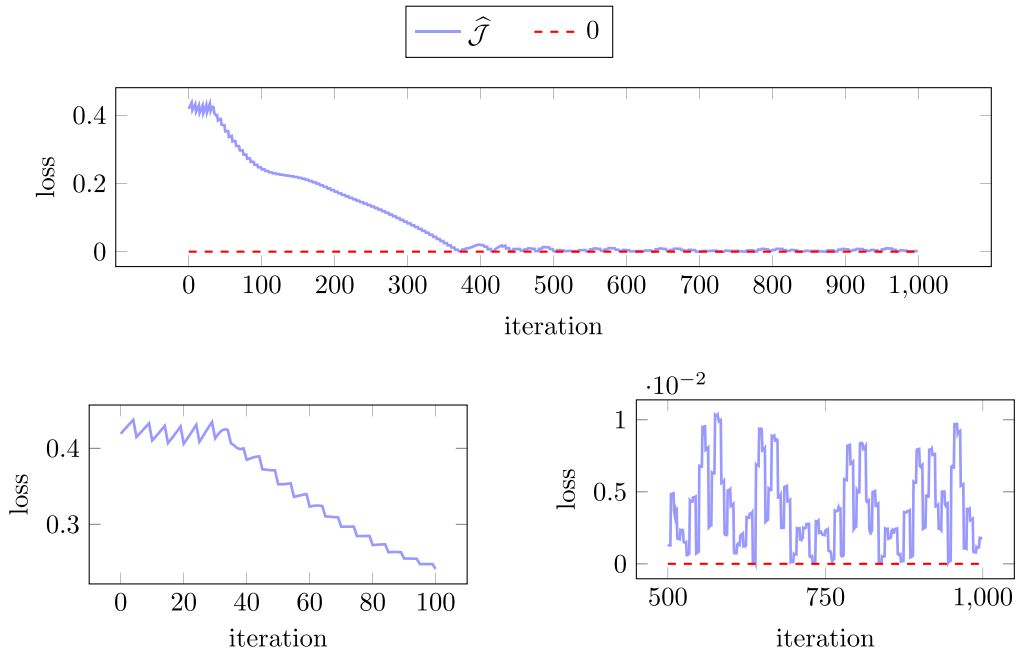
**Fig. 6.** Loss evolution during the WANs training for problem (19) with analytic solution $u^* = x(x - 1)$.
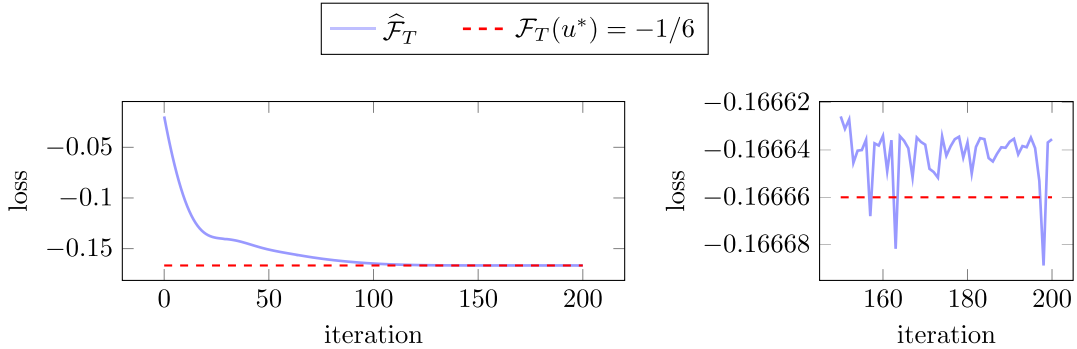


**Fig. 7.** Loss evolution during the DRM training in problem (19) with analytic solution $u^* = x(x - 1)$.

## 5.2. Comparison of WANs, the DRM, and the $D^2RM$ on smooth and singular problems

Now, we focus on the evolution of the relative error for the previous weak formulation of (19), but selecting the source so that the solution varies according to a parameter:

$$u_\alpha^* = x^\alpha (x - 1) \in H_0^1(0, 1), \qquad \alpha > 1/2. \tag{27}$$

### 5.2.1. Without singularities: $\alpha \geq 1$

We experiment individually for $\alpha \in \{2, 5, 10\}$ with 5000 training iterations for $u_{\text{NN}}$ in WANs, the DRM, and the $D^2RM$, which corresponds to a total of 25,000 iterations for WANs and the $D^2RM$, when taking into account the iterations dedicated to the test maximizers. Table 2 displays the relative errors along different stages of the training, and Fig. 9 shows the trial network predictions and error functions at the end of training.
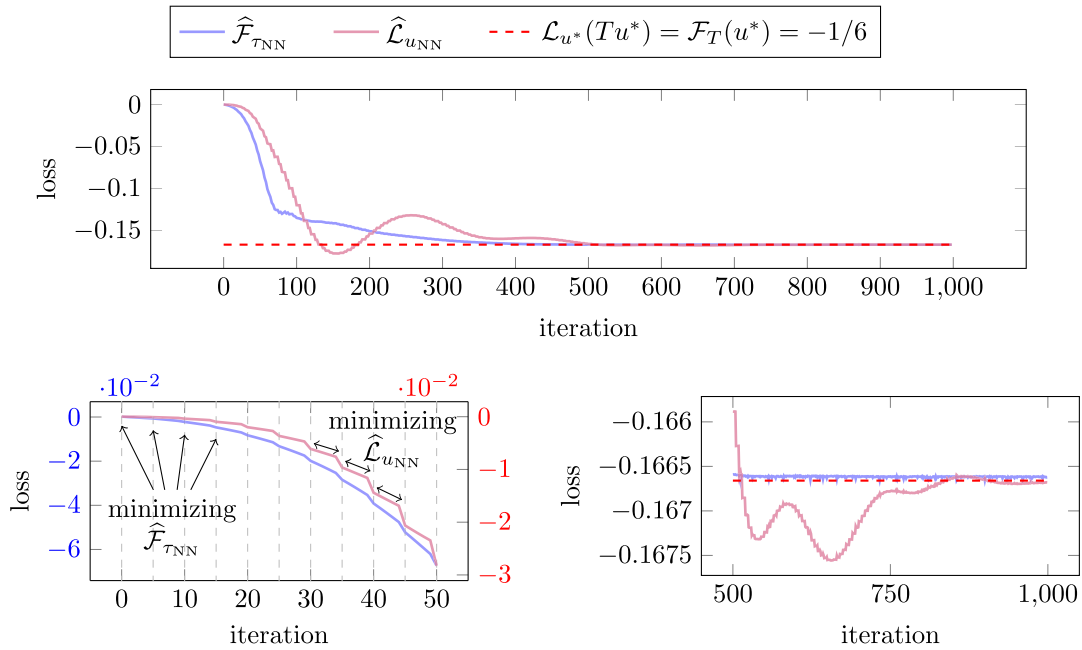
**Fig. 8.** Loss evolution of D²RM training for problem (19) with analytic solution $u^* = x(x-1)$.

**Table 2**
Relative errors of $u_{NN}$ (in WANs, the DRM, and the D²RM) and $\tau_{NN}(u_{NN})$ (in the D²RM) along different stages of the training progress in problem (19) with analytic solution $u_\alpha^* = x^\alpha(x-1)$ and $\alpha \in \{2, 5, 10\}$.

| Training progress | | 4% | 20% | 40% | 60% | 100% |
|---|---|---|---|---|---|---|
| Method | $\alpha$ | $\frac{\|u_{NN}-u^*\|_{\mathbb{U}}}{\|u^*\|_{\mathbb{U}}} \times 100$ | | | | |
| WANs | 2 | 2.49% | 3.43% | 5.92% | 7.40% | 10.07% |
| | 5 | 58.69% | 41.78% | 63.03% | 74.06% | 40.33% |
| | 10 | 93.15% | 93.95% | 89.04% | 68.85% | 367.61% |
| DRM | 2 | 3.40% | 2.47% | 1.17% | 0.30% | 0.23% |
| | 5 | 50.50% | 8.53% | 2.83% | 2.27% | 1.59% |
| | 10 | 58.55% | 9.51% | 3.39% | 2.83% | 1.69% |
| D²RM | 2 | 3.27% | 1.84% | 0.31% | 0.27% | 0.54% |
| | 5 | 59.13% | 12.93% | 2.92% | 2.52% | 1.56% |
| | 10 | 82.31% | 20.18% | 6.24% | 2.68% | 2.60% |
| Method | $\alpha$ | $\frac{\|\tau_{NN}(u_{NN})-Tu^*\|_{\mathbb{V}}}{\|Tu^*\|_{\mathbb{V}}} \times 100$ | | | | |
| D²RM | 2 | 3.36% | 1.94% | 0.48% | 0.45% | 0.58% |
| | 5 | 59.13% | 12.93% | 2.90% | 2.11% | 1.55% |
| | 10 | 83.51% | 20.18% | 6.23% | 2.68% | 2.61% |

WANs show poor results in all the cases, with a clear non-convergent tendency as the training progresses, possibly justified by the unstable behavior of the method at the continuous level. Thus, we discard the WANs for the remainder of the experiments and focus on the other two methods.

In the DRM and the D²RM, we observe a decreasing behavior of the relative error during training. We highlight the nearly identical behavior of the relative norm errors of $u_{NN}$ and $\tau_{NN}(u_{NN})$ in the D²RM, which suggests that the D²RM behaves as the DRM, as desired.
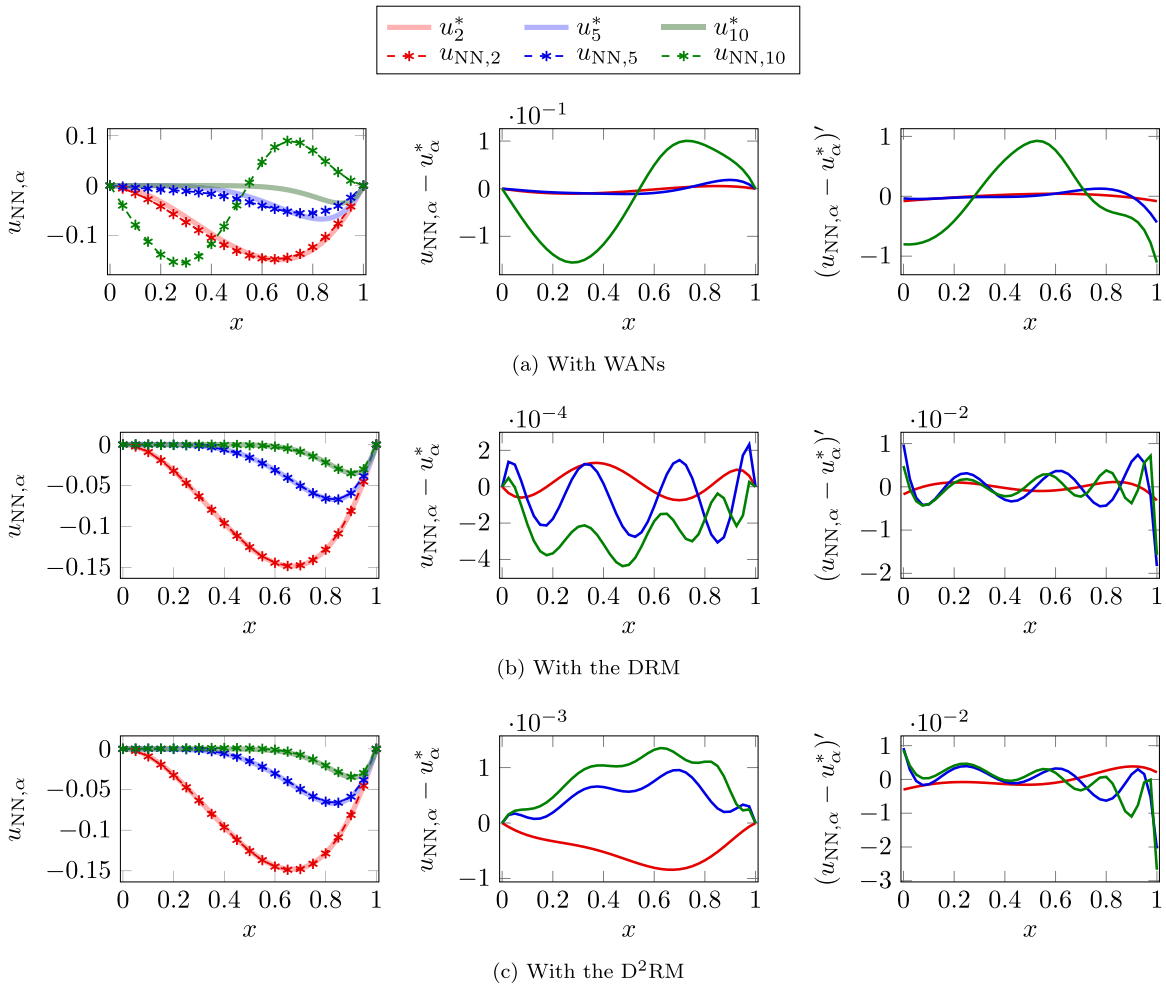
13

**Fig. 9.** Trial network predictions and errors in WANs, the DRM, and the D²RM in problem (19) with analytic solution $u_\alpha^* = x^\alpha(x-1)$ for $\alpha \in \{2, 5, 10\}$.

### 5.2.2. With singularities: $1/2 < \alpha < 1$

Here, $(u_\alpha^*)'(x) \to -\infty$ as $x \to 0$, which suggests that a large portion of the integration nodes should concentrate on a neighborhood of zero to appropriately capture the explosive trend of the derivative at that point. To this end, we divide the batch as the union of two equal-sized different $\beta(a, b)$ samples: one with $a = 1 = b$, and the other with $a = 10^4$ and $b = 1$. We individually experiment for $\alpha \in \{0.6, 0.7, 0.8\}$ with 100,000 iterations for approximating $u_{NN}$, which implies 500,000 for the D²RM when taking into account the iterations dedicated to approximate the trial-to-test operator. Table 3 displays the record of the relative error estimates along different stages of the training, and Fig. 10 shows the $u_{NN}$ predictions and error functions at the end of the training.

The DRM and D²RM decrease the relative errors at similar rates, slowing down notably from the 40% of the training progress onwards. For $\alpha \in \{0.7, 0.8\}$, we achieve final relative errors below 7.5% at the end of training, with loss predictions around $-0.3632$ and $-0.2562$ at the end of the training where the optimal values are around $-0.3646$ and $-0.2564$, respectively. For $\alpha = 0.6$, the relative error is above 20%, with a final loss prediction around $-0.6429$ where the optimal value is around $-0.6818$. We repeated several experiments in this case modifying the $a$ parameter of the beta distribution with the intention of concentrating more points in the singularity, but without success.

**Table 3**

Relative errors of $u_{\mathrm{NN}}$ (in the DRM, and the D$^2$RM) and $\tau_{\mathrm{NN}}(u_{\mathrm{NN}})$ (in the D$^2$RM) along different stages of the training progress in problem (19) with analytic solution $u_\alpha^* = x^\alpha(x-1)$ and $\alpha \in \{0.6, 0.7, 0.8\}$.

| Training progress | | 4% | 20% | 40% | 60% | 100% |
|---|---|---|---|---|---|---|
| Method | $\alpha$ | | | $\frac{\|u_{\mathrm{NN}}-u^*\|_{\mathbb{U}}}{\|u^*\|_{\mathbb{U}}} \times 100$ | | |
| DRM | 0.6 | 42.23% | 30.29% | 26.91% | 25.34% | 23.84% |
| | 0.7 | 18.47% | 9.49% | 7.42% | 6.65% | 5.95% |
| | 0.8 | 8.78% | 3.50% | 2.46% | 2.10% | 1.81% |
| D$^2$RM | 0.6 | 41.67% | 30.25% | 27.05% | 25.76% | 24.40% |
| | 0.7 | 14.12% | 9.99% | 7.91% | 6.98% | 6.15% |
| | 0.8 | 6.00% | 3.62% | 2.78% | 2.55% | 2.13% |
| Method | $\alpha$ | | | $\frac{\|\tau_{\mathrm{NN}}(u_{\mathrm{NN}})-Tu^*\|_{\mathbb{V}}}{\|Tu^*\|_{\mathbb{V}}} \times 100$ | | |
| D$^2$RM | 0.6 | 41.68% | 30.25% | 27.05% | 25.77% | 24.40% |
| | 0.7 | 14.12% | 10.00% | 7.91% | 6.98% | 6.15% |
| | 0.8 | 6.01% | 3.62% | 2.78% | 2.54% | 2.13% |

**Table 4**

Relative errors of $u_{\mathrm{NN}}$ (in the DRM, and the D$^2$RM) and $\tau_{\mathrm{NN}}(u_{\mathrm{NN}})$ (in the D$^2$RM) along different stages of the training progress in problem (19) with analytic solution (28).

| Training progress | 4% | 20% | 40% | 60% | 100% |
|---|---|---|---|---|---|
| Method | | | $\frac{\|u_{\mathrm{NN}}-u^*\|_{\mathbb{U}}}{\|u^*\|_{\mathbb{U}}} \times 100$ | | |
| DRM | 32.32% | 10.10% | 6.22% | 5.08% | 4.34% |
| D$^2$RM | 32.43% | 13.95% | 10.87% | 9.41% | 7.95% |
| Method | | | $\frac{\|\tau_{\mathrm{NN}}(u_{\mathrm{NN}})-Tu^*\|_{\mathbb{V}}}{\|Tu^*\|_{\mathbb{V}}} \times 100$ | | |
| D$^2$RM | 32.68% | 14.09% | 10.84% | 9.53% | 8.16% |

### 5.3. Pure diffusion equation with Dirac delta source

We now select the source $l = 4\delta_{1/2} \in H^{-1}(0,1) \setminus L^2(0,1)$, where $\delta_{1/2}$ denotes the Dirac delta at $1/2$, i.e., $\delta_{1/2}(v) = v(1/2)$ for all $v \in H_0^1(0,1)$. The analytic solution is

$$u^* = \begin{cases} 2x, & \text{if } x < 1/2, \\ 2(1-x), & \text{if } x > 1/2. \end{cases} \tag{28}$$

We train $u_{\mathrm{NN}}$ for 20,000 iterations selecting $a = 1 = b$ and $a = 10 = b$ in the $\beta(a,b)$ probability distribution for the two-sampled batch. Fig. 11 shows the trial network predictions at three different phases of the training progress, and Fig. 12 shows the error functions and their derivatives at those checkpoints. Table 4 displays a record of the relative errors. In this occasion, the D$^2$RM shows higher errors than the DRM.

Fig. 13 shows the losses evolution for the DRM and the D$^2$RM. We observe that in the DRM and the D$^2$RM, the loss related to the trial networks attain thresholds below $-1.98$, where $-2$ is the optimum at the continuous level, but the loss related to the optimal test networks remains much higher.

### 5.4. Pure convection equation in ultraweak form

We consider the spatial convection equation

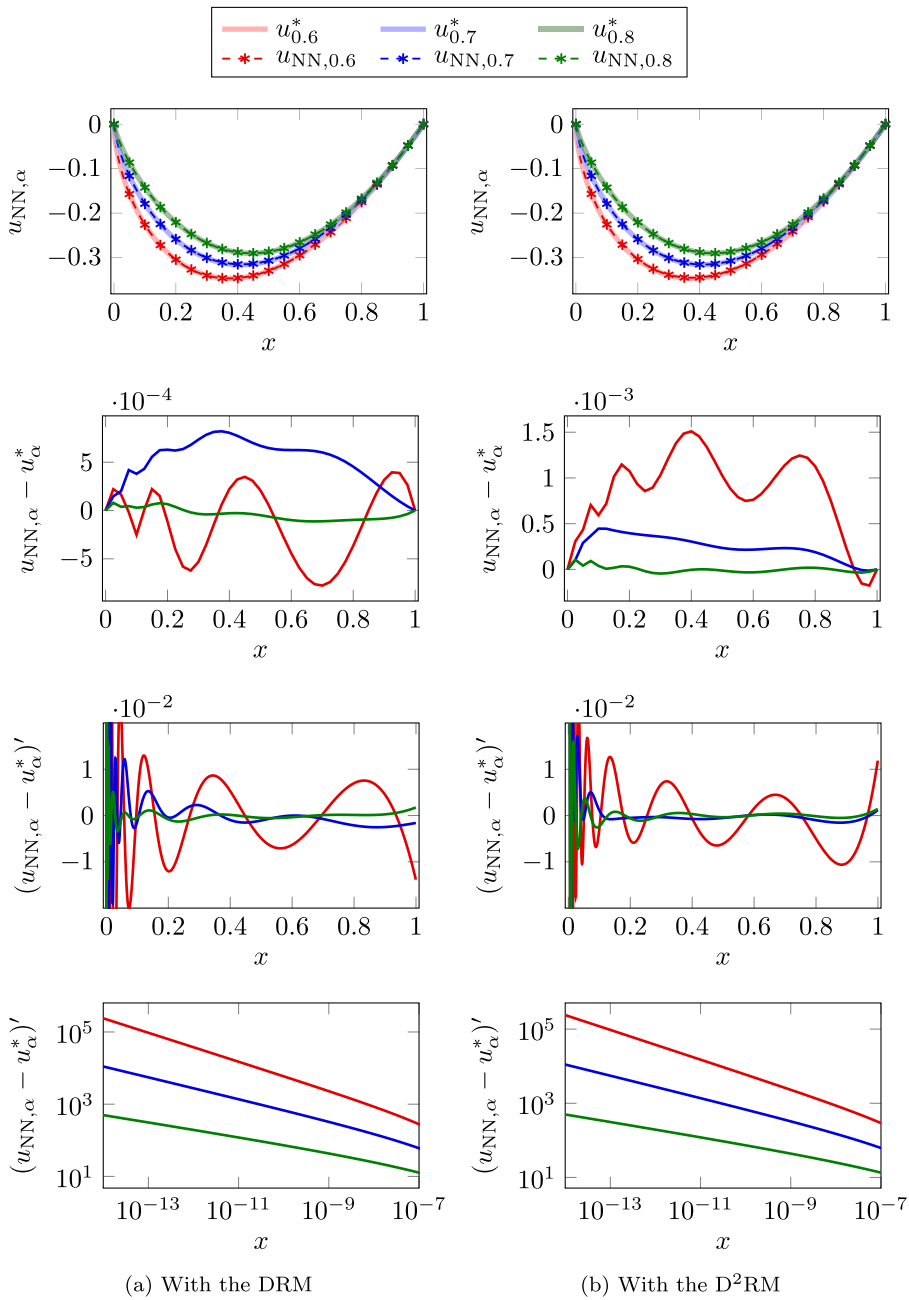$$\begin{cases} u' = \delta_{1/2}, & \text{in } (0,1), \\ u(0) = 0. \end{cases} \tag{29}$$

15

**Fig. 10.** Trial network predictions and error functions in the DRM and the D$^2$RM in problem (19) with analytic solution $u_\alpha^* = x^\alpha(x-1)$ for $\alpha \in \{0.6, 0.7, 0.8\}$. The last row is an augmented version of the third one in a neighborhood of zero.

Here, the (ultra)weak formulation is appropriate because $\delta_{1/2}$ does not belong to $L^2(0,1)$. Integration by parts yields:

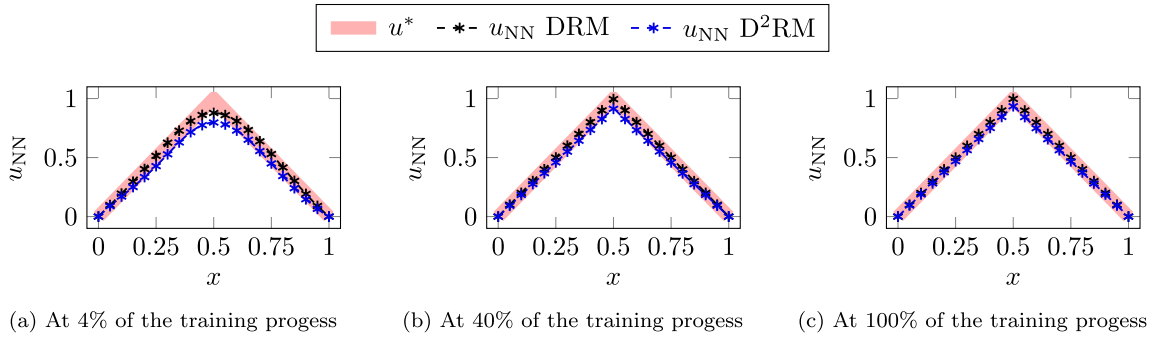$$b(u,v) := -\int_0^1 u v', \quad l(v) := v(1/2), \qquad u \in \mathbb{U}, v \in \mathbb{V}, \tag{30}$$

(a) At 4% of the training progress    (b) At 40% of the training progess    (c) At 100% of the training progess

**Fig. 11.** Trial network predictions for the DRM and the D$^2$RM in problem (19) with analytic solution (28).



(a) At 4% of the training progess    (b) At 40% of the training progess    (c) At 100% of the training progess
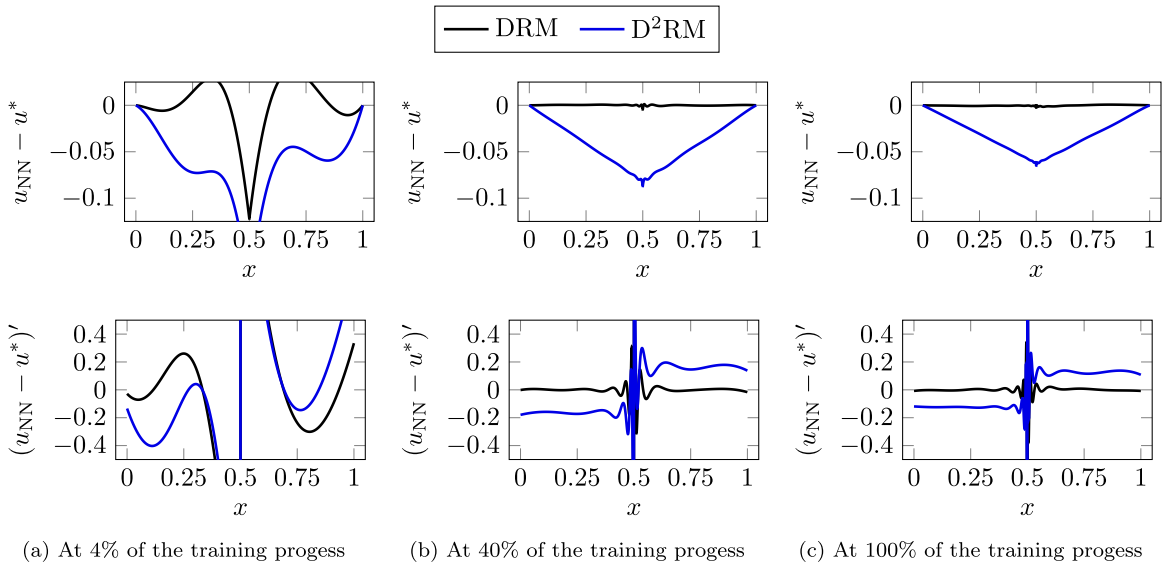
**Fig. 12.** Trial error functions in problem (19) with analytic solution (28) at different stages of the training progress.

with $\mathbb{U} := L^2(0, 1)$ and $\mathbb{V} := H_{0)}^1 := \{v \in H^1(0, 1) : v(1) = 0\}$. The trial-to-test operator is no longer the identity and has the following integral form [54,55]:

$$(Tu)(x) = \int_x^1 u(s)ds, \qquad u \in L^2(0, 1). \tag{31}$$

For the exact solution and its corresponding optimal test function, we have

$$u^* = \begin{cases} 0, & \text{if } 0 < x < 1/2, \\ 1, & \text{if } 1/2 < x < 1, \end{cases} \in L^2(0, 1), \qquad Tu^* = \begin{cases} 1/2 & \text{if } 0 < x < 1/2, \\ 1 - x, & \text{if } 1/2 < x < 1, \end{cases} \in H_{0)}^1(0, 1). \tag{32}$$

In our context of NNs, considering $T$ as an available operator is challenging, so we discard employing the GDRM in here, and alternatively employ the (DRM)$'$ and the D$^2$RM:

- **Adjoint Deep Ritz Method.** Following Section 2.4, item (c), we minimize $\mathcal{F}'$ to find an approximation to the optimal test function of the trial solution, i.e.,

$$Tu^* \approx \arg \min_{v_{\text{NN}} \in \mathbb{V}_{\text{NN}}} \mathcal{F}'(v_{\text{NN}}). \tag{33}$$

Later, we post-process the above minimizer by applying the available adjoint operator $A' = -d/dx$.
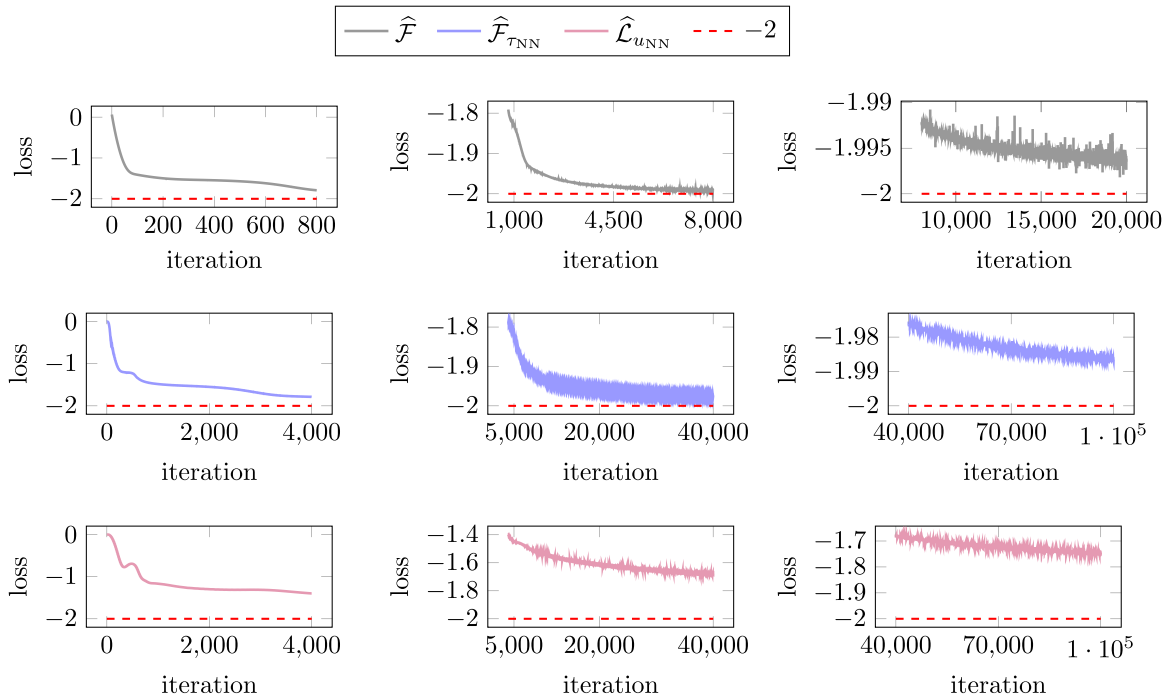
**Fig. 13.** Loss functions evolution in the DRM and the D²RM in problem (19) with exact solution (28).

**Table 5**

Relative errors of $v_{NN}$ and $A'v_{NN}$ in problem (29) with exact trial and optimal test solutions (32) along different stages of the training progress.

| Training progress | 4% | 20% | 40% | 60% | 100% |
|---|---|---|---|---|---|
| $\frac{\|v_{NN}-Tu^*\|_{\mathbb{V}}}{\|Tu^*\|_{\mathbb{V}}} \times 100$ | 19.03% | 4.58% | 3.55% | 3.20% | 2.83% |
| $\frac{\|A'v_{NN}-u^*\|_{\mathbb{U}}}{\|u^*\|_{\mathbb{U}}} \times 100$ | 21.95% | 5.28% | 4.10% | 3.69% | 3.27% |

We perform 50,000 training iterations. Table 5 records the evolution of the approximated relative norm errors $\frac{\|v_{NN}-Tu\|_{\mathbb{V}}}{\|Tu\|_{\mathbb{V}}}$ and $\frac{\|A'v_{NN}-u\|_{\mathbb{U}}}{\|u\|_{\mathbb{U}}}$ along the training progress. Fig. 14 shows the predictions and errors of both $v_{NN}$ and $A'v_{NN}$ along different stages of the training.

- **Deep Double Ritz Method.** We impose the outflow-boundary condition only to $\tau_{NN}$, letting $u_{NN}$ be boundary-free, and perform 50,000 iterations for $u_{NN}$ and increase the number of inner-loop iterations from four to nine (i.e., a total of 500,000 iterations if we account those dedicated to the test maximizer). Table 6 records the relative errors for $u_{NN}$ and $\tau_{NN}(u_{NN})$ along different stages of the training. Fig. 15 shows the trial and optimal test network predictions and error functions at the end of the training.

## 5.5. 2D pure convection in strong variational form

Let

$$\begin{cases} \dfrac{\partial u}{\partial x} + \dfrac{\partial u}{\partial y} = k\pi \sin\left(k\pi(x+y)\right), & \text{in } \Omega = (0,1) \times (0,1), \\ u(x,0) = u(0,y) = 0, & 0 \le x, y \le 1, \end{cases} \tag{34}$$
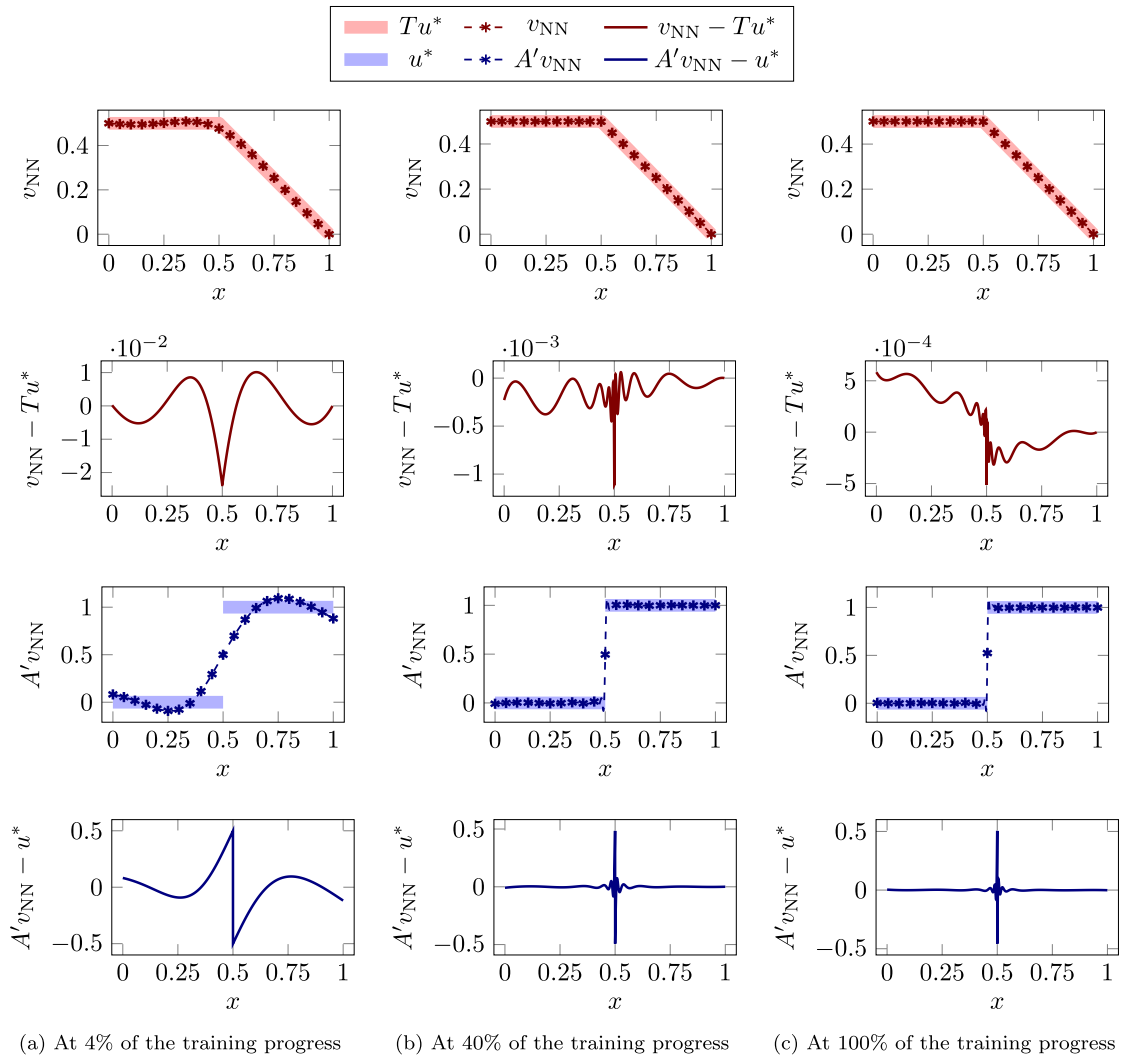
**Fig. 14.** $v_{\mathrm{NN}}$ network predictions, post-processings $A'v_{\mathrm{NN}}$, and corresponding error functions for the (DRM)$'$ in problem (29) with exact trial solution (32) along different stages of the training progress.

**Table 6**
Relative errors of $u_{\mathrm{NN}}$ and $\tau_n u_{\mathrm{NN}}$, respectively, along different stages of the training progress in problem (29) with exact trial and optimal test solutions (32).

| Training progress | 4% | 20% | 40% | 60% | 100% |
|---|---|---|---|---|---|
| $\frac{\|u_{\mathrm{NN}} - u^*\|_{\mathbb{U}}}{\|u^*\|_{\mathbb{U}}}$ | 42.05% | 29.69% | 18.42% | 11.20% | 8.93% |
| $\frac{\|\tau_{\mathrm{NN}}(u_{\mathrm{NN}}) - Tu^*\|_{\mathbb{V}}}{\|Tu^*\|_{\mathbb{V}}}$ | 37.27% | 28.25% | 19.02% | 9.55% | 6.13% |

with $k = 3/2$, and consider its strong variational formulation, i.e.,

$$b(u, v) := \int_\Omega \left( \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} \right) v, \quad l(v) := k\pi \int_\Omega \sin(k\pi(x + y))v, \quad u \in \mathbb{U}, v \in \mathbb{V}, \tag{35}$$

with $\mathbb{U} := \{u \in L^2(\Omega) : \partial u/\partial x + \partial u/\partial y \in L^2(\Omega), u(x, 0) = u(0, y) = 0, 0 \leq x, y \leq 1\}$ and $\mathbb{V} := L^2(\Omega)$. Its analytic solution is $u^* = \sin(3\pi x/2) \sin(3\pi y/2)$, and the trial-to-test operator is the PDE operator.
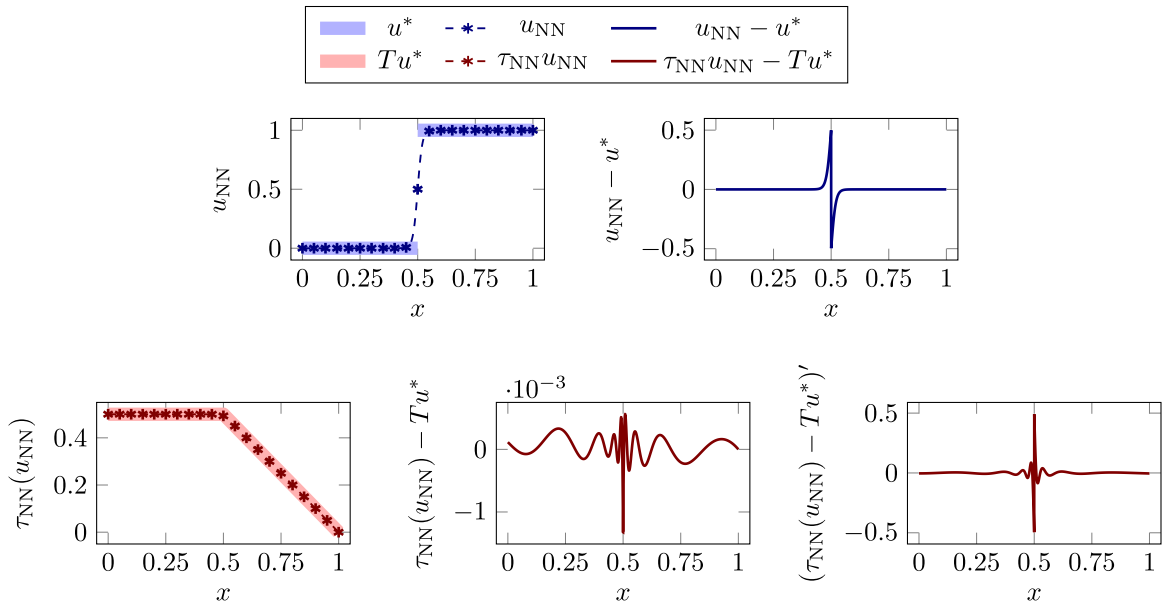
**Fig. 15.** $u_{\mathrm{NN}}$ and $\tau_{\mathrm{NN}}(u_{\mathrm{NN}})$ predictions, errors, and derivative of the errors for the D$^2$RM in problem (29) with exact trial and optimal test solutions (32).

We perform 200,000 iterations in the D$^2$RM with a training regime of nine iterations in the inner loop for each iteration in the outer loop. Exceptionally, we run the inner loop for 2000 iterations before the first outer-loop iteration. For integration, we consider 50 nodes on each axis (i.e., 2500 integration points on the entire domain due to the cartesian-product structure). We increase the NN architecture to three layers of 50-neuron width.

Fig. 16 shows the trial and optimal test predictions with corresponding error functions at the end of the training. The resulting relative errors are 3.62% and 1.70% for $u_{\mathrm{NN}}$ and $\tau_{\mathrm{NN}}(u_{\mathrm{NN}})$, respectively.

## 6. Conclusions and future work

We studied the problem of residual minimization for solving PDEs using Neural Networks. First, we considered a min–max saddle point problem coming from the definition of the dual norm as a maximum over the test space. This method turned out numerically unstable because of the non-Lipschitz continuity of the test maximizers. To overcome this, we rewrote the general residual minimization as a minimization of the Ritz functional employing optimal test functions. To carry out this alternative minimization while computing the optimal test functions over general problems, we proposed a Deep Double Ritz Method (D$^2$RM) that combines two nested Ritz optimization loops. This novel method constructs local approximations of the trial-to-test operator to express the optimal test functions as dependent on the trial functions. By doing this, NNs allowed us to easily combine the two nested Ritz problems in a way that is difficult to treat with traditional numerical methods.

We tested the D$^2$RM in several smooth and singular problems. Numerical results illustrate the advantages and limitations of the proposed methods. Among the advantages, we encounter the good approximation capabilities of NNs, the generality of the proposed method, and its implementation to different linear PDEs, variational formulations, and spatial dimensions. As main limitations, we face two common difficulties shared by most NN-based PDE solvers, namely: (a) lack of efficient integration methods (especially in presence of singular solutions where Monte Carlo methods fail), and (b) lack of efficient optimizers, which often fall in a local minima whose distance to global minima is uncertain. Other than that, numerical results look promising and are supported on a solid mathematical framework at the continuous level.

As future work, we will better analyze numerical integration techniques in the context of NNs; specifically, focused on Ritz-type minimizations. In addition, we will investigate the behavior of optimal test functions to propose enhanced stopping criteria in the inner loop. In particular, we believe that a global approximation of the trial-to-test operator with NNs would be helpful for this purpose.
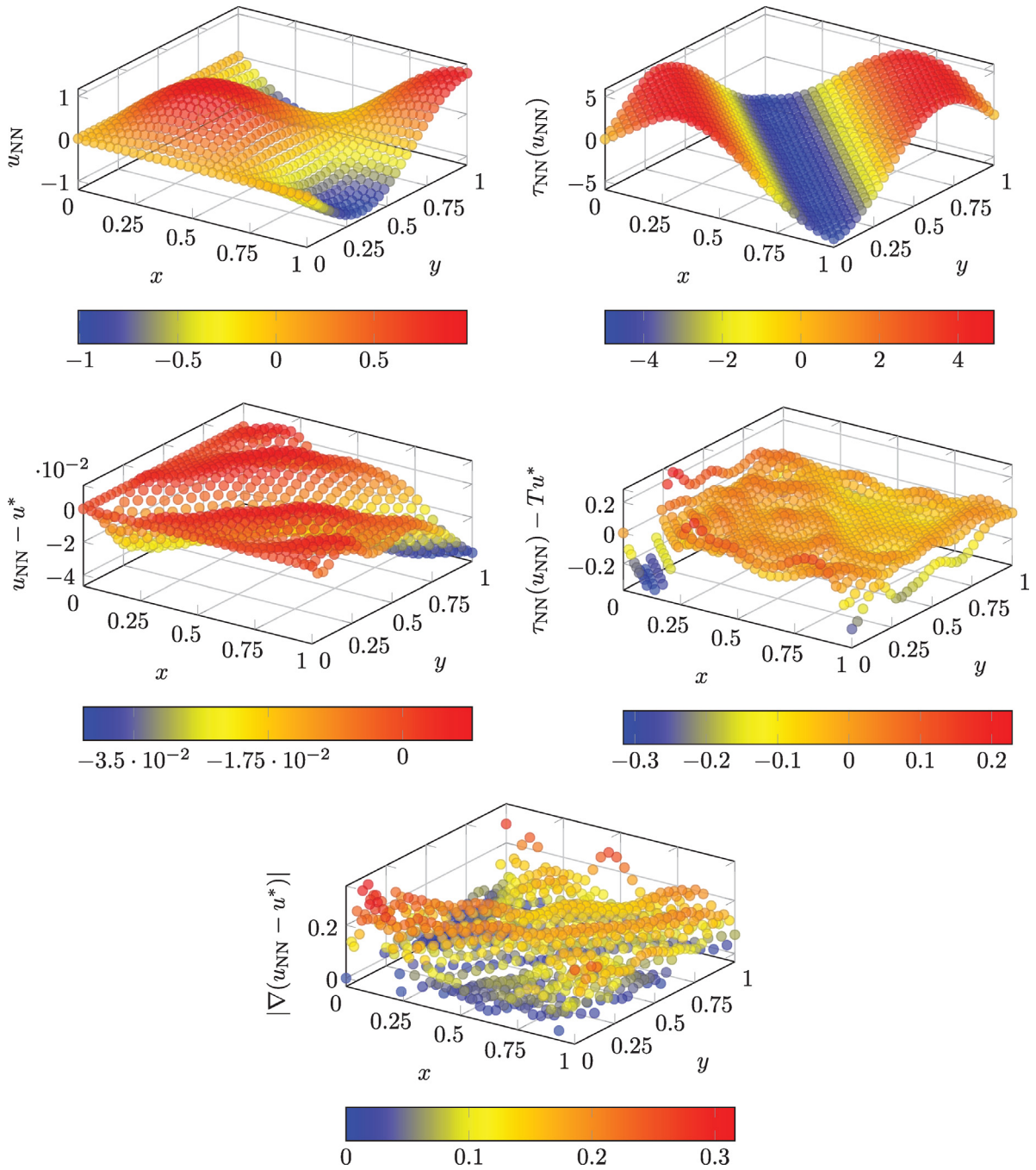
Fig. 16. $u_{\mathrm{NN}}$ and $\tau_{\mathrm{NN}}(u_{\mathrm{NN}})$ predictions and errors for the D$^2$RM in problem (34) with exact trial and optimal test solutions $u^* = \sin(3\pi x/2)\sin(3\pi y/2)$ and $Tu^* = 3\pi/2\sin(3\pi x/2)\sin(3\pi y/2)$, respectively.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

**Data availability**

No data was used for the research described in the article.

**Acknowledgments**

**Appendix. Numerical instability of the min–max approach**

Let $J : \mathbb{U} \setminus \{u^*\} \longrightarrow \mathbb{V}$ be the mapping that for each trial function returns the test maximizer of the actions of the residual $Bu - l \in \mathbb{V}'$ over the unitary sphere, i.e.,

$$J(u) := \arg \max_{\|v\|_{\mathbb{V}}=1} (Bu - l)(v). \tag{A.1}$$

Notice that $J$ is not well defined at $u^*$ because $Bu^* - l$ is the null operator.

**Proposition 1.** $J(u) \in \mathbb{V}$ is unique and

$$J(u) = \frac{T(u - u^*)}{\|T(u - u^*)\|_{\mathbb{V}}}, \qquad u \in \mathbb{U} \setminus \{u^*\}. \tag{A.2}$$

**Proof.** By the Riesz Representation Theorem, for $Bu - l \in \mathbb{V}'$, there exists a unique $r_u \in \mathbb{V}$ such that

$$(r_u, v)_{\mathbb{V}} = (Bu - l)(v), \ \forall v \in \mathbb{V} \qquad \text{and} \qquad \|r_u\|_{\mathbb{V}} = \|Bu - l\|_{\mathbb{V}'} = \max_{\|v\|_{\mathbb{V}}=1} (Bu - l)(v). \tag{A.3}$$

Let $w \in \mathbb{V}$ such that $\|w\|_{\mathbb{V}} = 1$ and

$$(Bu - l)(w) = \max_{\|v\|_{\mathbb{V}}=1} (Bu - l)(v). \tag{A.4}$$

By the Cauchy–Schwarz inequality:

$$0 \le \|Bu - l\|_{\mathbb{V}'} = (Bu - l)(w) = (r_u, w)_{\mathbb{V}} \le \|r_u\|_{\mathbb{V}}\|w\|_{\mathbb{V}} = \|r_u\|_{\mathbb{V}}, \tag{A.5}$$

where the equality holds if and only if $w = \lambda r_u$ for some $\lambda \ge 0$. Then, $w = \frac{r_u}{\|r_u\|_{\mathbb{V}}}$ is the unique solution. Employing Eq. (11), we find (A.2). $\square$

**Proposition 2.** $J$ is not Lipschitz continuous around $u^*$, i.e. there does not exist a constant $C > 0$ such that

$$\|J(u_1) - J(u_2)\|_{\mathbb{V}} \le C\|u_1 - u_2\|_{\mathbb{U}}, \qquad \forall u_1, u_2 \in \mathbb{U} \setminus \{u^*\}. \tag{A.6}$$

**Proof.** Assume by contradiction that there exists $C > 0$ such that (A.6) holds, and select $u_2 = 2u^* - u_1$ with $u_1 \ne u^*$. Then,

$$2 = \|J(u_1) - J(u_2)\|_{\mathbb{V}} \le C\|u_1 - u_2\|_{\mathbb{U}} = 2C\|u_1 - u^*\|_{\mathbb{U}}. \tag{A.7}$$

Letting $u_1 \to u^*$, we obtain $C \to \infty$. $\square$

# References

[1] Z. Zhang, Y. Wang, P.K. Jimack, H. Wang, MeshingNet: A new mesh generation method based on deep learning, in: International Conference on Computational Science, Springer, 2020, pp. 186–198.

[2] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, P.W. Battaglia, Learning mesh-based simulation with graph networks, 2020, arXiv preprint arXiv:2010.03409.

[3] M. Paszyński, R. Grzeszczuk, D. Pardo, L. Demkowicz, Deep learning driven self-adaptive hp finite element method, in: International Conference on Computational Science, Springer, 2021, pp. 114–121.

[4] T. Sluzalec, R. Grzeszczuk, S. Rojas, W. Dzwinel, M. Paszynski, Quasi-optimal $hp$-finite element refinements towards singularities via deep neural network prediction, 2022, arXiv preprint arXiv:2209.05844.

[5] C. Uriarte, D. Pardo, Á.J. Omella, A finite element based deep learning solver for parametric PDEs, Comput. Methods Appl. Mech. Engrg. 391 (2022) 114562.

[6] Á.J. Omella, D. Pardo, $r$-Adaptive deep learning method for solving partial differential equations, 2022, arXiv preprint arXiv:2210.10900.

[7] I. Brevis, I. Muga, K.G. van der Zee, A machine-learning minimal-residual (ML-MRes) framework for goal-oriented finite element discretizations, Comput. Math. Appl. 95 (2021) 186–199.

[8] I. Brevis, I. Muga, K.G. van der Zee, Neural control of discrete weak formulations: Galerkin, least squares & minimal-residual methods with quasi-optimal weights, Comput. Methods Appl. Mech. Engrg. (2022) 115716.

[9] 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, J. Comput. Phys. 378 (2019) 686–707.

[10] S.-M. Qin, M. Li, T. Xu, S.-Q. Dong, RAR-PINN algorithm for the data-driven vector-soliton solutions and parameter discovery of coupled nonlinear equations, 2022, arXiv preprint arXiv:2205.10230.

[11] E. Weinan, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (1) (2018) 1–12.

[12] J.M. Taylor, D. Pardo, I. Muga, A deep Fourier residual method for solving PDEs using neural networks, 2022, arXiv preprint arXiv:2210.14129.

[13] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.

[14] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, 2019, arXiv preprint arXiv:1912.00873.

[15] R. Khodayi-Mehr, M. Zavlanos, VarNet: Variational neural networks for the solution of partial differential equations, in: Learning for Dynamics and Control, PMLR, 2020, pp. 298–307.

[16] E. Kharazmi, Z. Zhang, G.E. Karniadakis, hp-VPINNs: Variational physics-informed neural networks with domain decomposition, Comput. Methods Appl. Mech. Engrg. 374 (2021) 113547.

[17] Y. Shang, F. Wang, J. Sun, Deep Petrov-Galerkin method for solving partial differential equations, 2022, arXiv preprint arXiv:2201.12995.

[18] Y. Shin, J. Darbon, G.E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs, 2020, arXiv preprint arXiv:2004.01806.

[19] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, M.W. Mahoney, Characterizing possible failure modes in physics-informed neural networks, Adv. Neural Inf. Process. Syst. 34 (2021) 26548–26560.

[20] S. Wang, X. Yu, P. Perdikaris, When and why PINNs fail to train: A neural tangent kernel perspective, J. Comput. Phys. 449 (2022) 110768.

[21] A. Daw, J. Bu, S. Wang, P. Perdikaris, A. Karpatne, Rethinking the importance of sampling in physics-informed neural networks, 2022, arXiv preprint arXiv:2207.02338.

[22] J. Bramble, R. Lazarov, J. Pasciak, A least-squares approach based on a discrete minus one inner product for first order systems, Math. Comp. 66 (219) (1997) 935–955.

[23] P.B. Bochev, M.D. Gunzburger, Least-Squares Finite Element Methods, Vol. 166, Springer Science & Business Media, 2009.

[24] V.M. Calo, M. Los, Q. Deng, I. Muga, M. Paszynski, Isogeometric residual minimization method (iGRM) with direction splitting preconditioner for stationary advection-dominated diffusion problems, Comput. Methods Appl. Mech. Engrg. 373 (2021) 113214.

[25] V.M. Calo, A. Ern, I. Muga, S. Rojas, An adaptive stabilized conforming finite element method via residual minimization on dual discontinuous Galerkin norms, Comput. Methods Appl. Mech. Engrg. 363 (2020) 112891.

[26] R.J. Cier, S. Rojas, V.M. Calo, Automatically adaptive, stabilized finite element method via residual minimization for heterogeneous, anisotropic advection–diffusion–reaction problems, Comput. Methods Appl. Mech. Engrg. 385 (2021) 114027.

[27] Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, J. Comput. Phys. 411 (2020) 109409.

[28] G. Bao, X. Ye, Y. Zang, H. Zhou, Numerical solution of inverse problems by weak adversarial networks, Inverse Problems 36 (11) (2020) 115003.

[29] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, Commun. ACM 63 (11) (2020) 139–144.

[30] L. Demkowicz, J. Gopalakrishnan, A class of discontinuous Petrov–Galerkin methods. II. Optimal test functions, Numer. Methods Partial Differential Equations 27 (1) (2011) 70–105.

[31] L.F. Demkowicz, J. Gopalakrishnan, An overview of the discontinuous Petrov Galerkin method, in: Recent Developments in Discontinuous Galerkin Finite Element Methods for Partial Differential Equations, Springer, 2014, pp. 149–180.

[32] A. Cohen, W. Dahmen, G. Welper, Adaptivity and variational stabilization for convection-diffusion equations, ESAIM Math. Model. Numer. Anal. 46 (5) (2012) 1247–1273.

[33] D. Broersen, W. Dahmen, R. Stevenson, On the stability of DPG formulations of transport equations, Math. Comp. 87 (311) (2018) 1051–1082.

[34] L. Demkowicz, J. Gopalakrishnan, B. Keith, The DPG-star method, Comput. Math. Appl. 79 (11) (2020) 3092–3116.

[35] P. Petersen, M. Raslan, F. Voigtlaender, Topological properties of the set of functions generated by neural networks of fixed size, Found. Comput. Math. 21 (2) (2021) 375–444.

[36] N. Lei, D. An, Y. Guo, K. Su, S. Liu, Z. Luo, S.-T. Yau, X. Gu, A geometric understanding of deep learning, Engineering 6 (3) (2020) 361–374.

[37] J. Brunken, K. Smetana, K. Urban, (Parametrized) first order transport equations: Realization of optimally stable Petrov–Galerkin methods, SIAM J. Sci. Comput. 41 (1) (2019) A592–A621.

[38] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.

[39] B.C. Csáji, et al., Approximation with artificial neural networks, Etvs Lornd University, Hungary, 2001, p. 7.

[40] A. Kratsios, I. Bilokopytov, Non-euclidean universal approximation, Adv. Neural Inf. Process. Syst. 33 (2020) 10635–10646.

[41] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Marchine Learn. Res. 18 (2018) 1–43.

[42] C.C. Margossian, A review of automatic differentiation and its efficient implementation, Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 9 (4) (2019) e1305.

[43] P.J. Davis, P. Rabinowitz, Methods of Numerical Integration, Courier Corporation, 2007.

[44] G. Leobacher, F. Pillichshammer, Introduction to Quasi-Monte Carlo Integration and Applications, Springer, 2014.

[45] J.A. Rivera, J.M. Taylor, Á.J. Omella, D. Pardo, On quadrature rules for solving partial differential equations using neural networks, Comput. Methods Appl. Mech. Engrg. 393 (2022) 114710.

[46] A.K. Gupta, S. Nadarajah, Handbook of Beta Distribution and Its Applications, CRC Press, 2004.

[47] S. Weinzierl, Introduction to Monte Carlo methods, 2000, arXiv Preprint Hep-Ph/0006269.

[48] S. Ruder, An overview of gradient descent optimization algorithms, 2016, arXiv preprint arXiv:1609.04747.

[49] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[50] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016, arXiv preprint arXiv:1603.04467.

[51] F.J.J. Joseph, S. Nonsiri, A. Monsakul, Keras and TensorFlow: A hands-on experience, in: Advanced Deep Learning for Engineers and Scientists, Springer, 2021, pp. 85–111.

[52] C. Meng, M. Sun, J. Yang, M. Qiu, Y. Gu, Training deeper models by GPU memory optimization on TensorFlow, in: Proc. of ML Systems Workshop in NIPS, Vol. 7, 2017.

[53] A. Sergeev, M. Del Balso, Horovod: fast and easy distributed deep learning in TensorFlow, 2018, arXiv preprint arXiv:1802.05799.

[54] J. Gopalakrishnan, Five lectures on DPG methods, 2013, arXiv preprint arXiv:1306.0557.

[55] J. Muñoz-Matute, D. Pardo, L. Demkowicz, Equivalence between the DPG method and the exponential integrators for linear parabolic problems, J. Comput. Phys. 429 (2021) 110016.