ARCHIVUM MATHEMATICUM (BRNO)
Tomus 59 (2023), 249–261

DEEP LEARNING FOR GRADIENT FLOWS USING THE BREZIS–EKELAND PRINCIPLE

Laura Carini, Max Jensen, and Robert Nürnberg

Abstract. We propose a deep learning method for the numerical solution of partial differential equations that arise as gradient flows. The method relies on the Brezis–Ekeland principle, which naturally defines an objective function to be minimized, and so is ideally suited for a machine learning approach using deep neural networks. We describe our approach in a general framework and illustrate the method with the help of an example implementation for the heat equation in space dimensions two to seven.

1. Introduction

In this paper we advocate a deep learning approach for solving parabolic partial differential equations (PDEs)

$$u_t + \partial\phi(u) = f\,,$$

that arise as evolution equations for gradient flows. We exploit the variational principle of the seminal papers by Brezis and Ekeland, [2,3], now commonly known as the Brezis–Ekeland principle, [17,19]. We also refer to [21] for related work, as well as to [7] for alternative variational formulations for large classes of PDEs.

Using neural networks for the numerical solution of PDEs has become increasingly popular over the last decade. An advantage of neural network-based approaches is their suitability for high-dimensional problems. For a comprehensive review of current developments we refer to [1,12]. Among currently popular techniques are methods based on residual minimization, e.g. see [15, 16, 18], and on the reformulation as a backward stochastic differential equation, e.g. [8,9,10,14].

Most relevant for this work are variational approaches. For an elliptic problem, E and Yu, [6], proposed a deep learning method based on a variational principle that leads to a natural optimization framework. An approach connecting variational principles with convex duality for stationary equations was taken in [11], which mirrors some aspects of our work for gradient flows.

The aim of our approach is three-fold:

2020 *Mathematics Subject Classification*: primary 35K15; secondary 35A15, 68T07.

Key words and phrases: machine learning, deep neural networks, gradient flows, Brezis–Ekeland principle, adversarial networks, differential equations.

Received September 28,2022, accepted December 1, 2022. Editor J. Chleboun.

DOI: 10.5817/AM2023-3-249

(1) Machine learning approaches have been criticized for their less developed methodology to bound, or at least estimate, the approximation error. It is therefore interesting that the minimum of the Brezis–Ekeland functional, which is minimized during the learning process, is guaranteed to be zero for the exact solution, thus providing an error measure that is known at the point of computing the neural net approximation.

(2) Adversarial networks have very successfully been applied across multiple problem classes of machine learning. We were intrigued by the question whether duality can be a context in which the concept of adversarial networks is translated to partial differential equations as well as to convex analysis, by introducing a neural network for the primal and another one for the dual problem. In a resulting min-max formulation the training stages of the respective networks take opposing, or adversarial, roles in finding the value of the joint loss functional.

(3) Finally, we wish to construct a method which takes advantage of the specific structural properties of gradient flows, based on the relevance of these properties in the literature for the construction of finite element methods for time-dependent PDEs.

The outline of the remainder of the paper is as follows: in section 2 we introduce gradient flows and the Brezis–Ekeland principle; in section 3 we formulate our deep learning approach; in section 4 we discuss the computer implementation of the method; in section 5 we present numerical experiments, followed by conclusions.

2. The Brezis–Ekeland principle for gradient flows

Let $V \subset H \subset V^*$ be a Gelfand triple and $T > 0$ a fixed time. In addition, let $\phi \in C^1(V)$ be convex, $f \in L^2(0,T;V^*)$ and $u_0 \in V$. We consider the gradient flow

$$u_t + \partial\phi(u) = f \quad \text{a.e. in } \ (0,T)\,, \quad u(0) = u_0\,. \tag{2.1}$$

The Brezis–Ekeland principle asserts that solutions $u \in Y$ to (2.1), with

$$Y := \{w \in L^2(0,T;V) \cap H^1(0,T;V^*) \ : \ w(0) = u_0\}\,,$$

are the global minimizers of the functional $\Phi : Y \to [0,\infty]$ defined by

$$\begin{aligned} \Phi(w) =& \tfrac{1}{2}\|w(T)\|_H^2 - \tfrac{1}{2}\|w(0)\|_H^2 + \int_0^T \phi(w) + \phi^*(f - w_t) - \langle f, w\rangle \ \mathrm{d}t \\ =& \int_0^T \phi(w) + \phi^*(f - w_t) + \langle w_t - f, w\rangle \ \mathrm{d}t\,, \end{aligned} \tag{2.2}$$

using $\int_0^T \langle w_t, w\rangle \ \mathrm{d}t = \frac{1}{2}\|w(T)\|_H^2 - \frac{1}{2}\|w(0)\|_H^2$. Here $\langle\cdot,\cdot\rangle$ is the duality pairing between V and V^*, and $\phi^*(w) = \sup_{v\in V}\langle w, v\rangle - \phi(v)$ is the conjugate of ϕ. In fact, owing to [17, Theorem 8.99], u solves (2.1) if and only if

$$\Phi(u) = \min\{\Phi(w) \ : \ w \in Y\} = 0\,. \tag{2.3}$$

Inspired by the work in [20], we now consider a time-discrete variant of (2.2) and the associated minimization problem. For that purpose, we divide $[0, T]$ into N sub-intervals with end points $t_0 = 0 < t_1 < \cdots < t_N = T$.

The parabolic nature of (2.1) ensures that $u(t)$ only depends on $u(s)$ if $s \leq t$, but not if $s > t$. Together with the Brezis–Ekeland principle (2.3) this guarantees that minimization and summation may be interchanged:

$$\begin{aligned}
(2.4)\quad & \min_{w \in Y} \Phi(w) \\
&= \min_{w \in Y} \sum_{n=1}^{N} \int_{t_{n-1}}^{t_n} \phi(w) + \phi^*(f - w_t) + \langle w_t - f, w \rangle \, \mathrm{d}t \\
&= \sum_{n=1}^{N} \min_{\substack{w_n \in Y \\ w_n(t_{n-1}) = w_{n-1}(t_{n-1})}} \int_{t_{n-1}}^{t_n} \phi(w_n) + \phi^*(f - (w_n)_t) + \langle (w_n)_t - f, w_n \rangle \, \mathrm{d}t,
\end{aligned}$$

where we have defined $w_0(0) = u_0$. The representation (2.4) of the minimization problem suggests a clear strategy for our deep learning method: we will sequentially solve N optimization problems for the PDE (2.1) on the time intervals $[t_{n-1}, t_n]$, where the initial data is either given by u_0 at the first step, or by the previously computed solution at time t_{n-1}.

The heat equation. The canonical example of a gradient flow is the heat equation. Given a Lipschitz domain $\Omega \subset \mathbb{R}^d$, $d \geq 1$, we consider the PDE:

$$(2.5)\qquad \begin{cases} u_t - \kappa \Delta u = f, & \text{in} \quad (0, T) \times \Omega\,, \\ u(0, \cdot) = u_0, & \text{in} \quad \Omega\,, \\ u = 0, & \text{on} \quad (0, T) \times \partial\Omega\,, \end{cases}$$

where $\kappa > 0$. Upon defining $H = L^2(\Omega)$, $V = H_0^1(\Omega)$ and $\phi(u) = \frac{\kappa}{2} \|\nabla u\|_{L^2}^2$, the problem (2.5) is a special case of (2.1), and the Brezis–Ekeland functional (2.2) in this case reduces to:

$$(2.6)\qquad \Phi(w) = \int_0^T \frac{\kappa}{2} \|\nabla w\|_{L^2}^2 + \frac{1}{2\kappa} \left[\sup_{v \neq 0} \frac{\langle f - w_t, v \rangle}{\|\nabla v\|_{L^2}} \right]^2 + \langle w_t - f, w \rangle \, \mathrm{d}t\,,$$

where we have used that $\phi^*(w) = \frac{1}{2\kappa} \left[\sup_{v \in H_0^1(\Omega) \setminus \{0\}} \frac{\langle w, v \rangle}{\|\nabla v\|_{L^2}} \right]^2$, e.g. see [17]. We remark that ϕ^* defines a norm on $H^{-1}(\Omega) = (H_0^1(\Omega))^*$. Combining (2.4) and (2.6) we find that $\min_{w \in Y} \Phi(w)$ equals

$$\begin{aligned}
& \sum_{n=1}^{N} \min_{\substack{w_n(t_{n-1}) \\ = w_{n-1}(t_{n-1})}} \int_{t_{n-1}}^{t_n} \frac{\kappa}{2} \|\nabla w_n\|_{L^2}^2 + \frac{1}{2\kappa} \left[\sup_{v \neq 0} \frac{\langle f - (w_n)_t, v \rangle}{\|\nabla v\|_{L^2}} \right]^2 \\
(2.7)\qquad & \qquad + \langle (w_n)_t - f, w_n \rangle \, \mathrm{d}t\,,
\end{aligned}$$

with the solution u of (2.5) being a minimizer in the sense that $\Phi(u) = 0$ and that the choice $w_n = u$, $n \in \{1, \ldots, N\}$, yields a minimizer of (2.7) over Y^N.

3. The deep learning approach

In this section we discuss a deep learning algorithm to find approximations of the solution to (2.1). For simplicity we restrict our attention to the heat equation, so that (2.7) is our starting point.

We wish to find the approximations $u_h^n \approx u(t_n, \cdot)$, $n = 1, \dots, N$, where u_h^n is given by a neural network. More generally, in this paper we view a neural network $\hat{u}_h$ as a function determined through its weights θ. Given θ the neural network takes a position $x \in \Omega$ as input and returns $\hat{u}_h(x; \theta)$ as output. The approximation set containing the functions u_h^n is thus given by

$$\mathbb{U}_h := \{\hat{u}_h(\cdot; \theta) \,:\, \theta \in \Theta\},$$

where Θ is the set of possible weights. In this notation $u_h^n = \hat{u}_h(\cdot; \theta)$ is the neural network with a choice of weights θ determined through the method described in this section.

In order to define a discrete version of the Brezis–Ekeland functional (2.7) and perform its minimization over $\mathbb{U}_h$, we turn our attention to the interpretation of v in $\phi^*(w) = \frac{1}{2\kappa}\left[\sup_{v \in H_0^1(\Omega)\setminus\{0\}} \langle w, v\rangle / \|\nabla v\|_{L^2}\right]^2$. Also the v are approximated by neural networks. Since their architecture may be different compared to $\hat{u}_h$, we introduce $\hat{v}_h(x; \eta)$ and

$$\mathbb{V}_h := \{\hat{v}_h(\cdot; \eta) \,:\, \eta \in \mathrm{H}\},$$

where H is the set of possible weights in $\hat{v}_h$.

While we assume $\mathbb{U}_h, \mathbb{V}_h \subset H^1(\Omega)$ throughout, elements of $\mathbb{U}_h$ and $\mathbb{V}_h$ will in general not belong to $H_0^1(\Omega)$ because the Dirichlet boundary conditions may not be satisfied homogeneously. Therefore, similarly to [6], we introduce a penalty term into $\phi^*(w)$ to obtain the functional

$$\phi_h^*(w_h) = \frac{1}{2\kappa}\left[\sup_{v_h \in \mathbb{V}_h\setminus\{0\}} \frac{(w_h, v_h)}{\left(\|\nabla v_h\|_{L^2}^2 + \lambda \|v_h\|_{L^2(\partial\Omega)}^2\right)^{\frac{1}{2}}}\right]^2, \tag{3.1}$$

where $\lambda > 0$ is a penalty parameter depending on the structure of the neural net. Then the denominator in (3.1) cannot vanish for $v_h \neq 0$ due to a Poincaré-Friedrichs inequality and the penalization weakly imposes homogeneous boundary conditions on any maximising v_h as $\lambda \to \infty$. We note that such penalization strategies to enforce Dirichlet boundary conditions are common in deep learning approaches, see e.g. [6], and have a long tradition in discontinuous Galerkin approximation methods, [5, §4.2].

It remains to discretize the time derivatives in the Brezis–Ekeland functional. To this end we substitute $(w_n)_t$ in (2.7) by backward time differences. Let $\Delta t_n = t_n - t_{n-1}$ and let $(\cdot, \cdot)$ denote the L^2-inner product over Ω. Inspired by (2.7), we then define the solution of the deep learning method through the following sequence of optimization problems: Given $u_h^{n-1} \in \mathbb{U}_h$, for $n = 1, \dots, N$, find a minimizer

$u_h^n \in \mathbb{U}_h$ to

$$\begin{aligned}\Phi_n(w_h) = \frac{\kappa\Delta t_n}{2}\|\nabla w_h\|_{L^2}^2 + \Delta t_n \phi_h^*\Big(f - \frac{w_h - u_h^{n-1}}{\Delta t_n}\Big) \\ + (w_h - u_h^{n-1}, w_h) + \lambda\|w_h\|_{L^2(\partial\Omega)}^2 ,\end{aligned} \tag{3.2}$$

where we have once again added a penalization term; this time to weakly impose homogeneous Dirichlet boundary conditions on w_h.

Obtaining the minimizer of (3.2) requires a maximization to evaluate $\phi_h^*(f - (w_h - u_h^{n-1})/\Delta t_n)$, see (3.1). For the remainder of this section we focus on an algorithm for solving this min-max problem. In the subsequent text it will be convenient to refer to

$$\widetilde{\phi}_h^*(w_h; v_h) = \frac{1}{2\kappa}\left[\frac{(w_h, v_h)}{(\|\nabla v_h\|_{L^2}^2 + \lambda\|v_h\|_{L^2(\partial\Omega)}^2)^{\frac{1}{2}}}\right]^2 ,$$

which is equal to $\phi_h^*(w_h)$ if v_h is a maximizer. Similarly, we write

$$\widetilde{\Phi}_n(w_h; p_h) = \frac{\kappa\Delta t_n}{2}\|\nabla w_h\|_{L^2}^2 + \Delta t_n\, p_h + (w_h - u_h^{n-1}, w_h) + \lambda\|w_h\|_{L^2(\partial\Omega)}^2 , \tag{3.3}$$

which equals $\Phi_n(w_h)$ upon choosing $p_h = \phi_h^*\big(f - \frac{w_h - u_h^{n-1}}{\Delta t_n}\big)$.

Algorithm 1

1: Compute an approximation $u_h^0 \in \mathbb{U}_h$ to u_0
2: **for** $n = 1, \ldots, N$ **do**
3: $\quad u_h^{n,0} \leftarrow u_h^{n-1}$, $k \leftarrow 0$
4: $\quad$ **while** termination_criterion($u_h^{n,k}$, k) = FALSE **do**
5: $\qquad k \leftarrow k + 1$
6: $\qquad p_h^{n,k} = \max_{v_h \in \mathbb{V}_h \setminus \{0\}} \widetilde{\phi}_h^*(f - (u_h^{n,k-1} - u_h^{n-1})/\Delta t_n; v_h)$
7: $\qquad u_h^{n,k} \in \arg\min_{w_h \in \mathbb{U}_h} \widetilde{\Phi}_n(w_h; p_h^{n,k})$
8: $\quad$ **end while**
9: $\quad u_h^n \leftarrow u_h^{n,k}$
10: **end for**

We shall base the minimization of (3.2) on Algorithm 1. The approximation of the initial conditions in line 1 of the algorithm is a supervised learning problem. Lines 2 and 10 frame the iteration over the time steps. Lines 4 and 8 implement a loop where the optimization of $\widetilde{\phi}_h^*$ (line 6) and $\widetilde{\Phi}_n$ (line 7) are alternated.

Remark 3.1. Alternatively to the above, one could use $\phi^*(w) = \frac{1}{2\kappa}\|\nabla\Delta^{-1}w\|_{L^2}^2$ for the formulation of the method, see [17, Example 8.104]. In this scenario we envisage $\Delta^{-1}u$ being approximated by a neural net, using an existing methodology for solving the Laplace problem.

4. BENNO: Brezis–Ekeland Neural Network Optimizer

We make a full Python implementation [4] of our deep learning approach available on Github, which is called *Brezis–Ekeland Neural Network Optimizer*, in short BENNO.

Neural Network Structure. We describe the internal structure of the neural networks $\hat{u}_h$ and $\hat{v}_h$, which were introduced in the previous section. Both these neural networks use five densely-connected layers with a linear activation function for the input and output layers and a (leaky) rectified linear unit activation function $\sigma(s) = \max\{0, s\} + \mu \min\{0, s\}$, $\mu \geq 0$, for the inner layers. Each layer is made of m nodes, except for the output layer that presents a single node. This means that the neural network $\hat{u}_h$ has the following architecture:

$$S^1(x) = W^1 x + b^1,\ S^2(x) = \sigma(W^2 S^1(x) + b^2)\,,\ S^3(x) = \sigma(W^3 S^2(x) + b^3)\,,$$
$$S^4(x) = \sigma(W^4 S^3(x) + b^4)\,,\ S^5(x) = W^5 S^4(x) + b^5, \quad \text{and} \quad \hat{u}_h(x;\theta) = S^5(x)\,,$$

where the set of parameters of the neural network $\hat{u}_h(\cdot;\theta)$ are given by

$$\theta = \{W^1, b^1, W^2, b^2, W^3, b^3, W^4, b^4, W^5, b^5\}\,,$$

with $W^1 \in \mathbb{R}^{m\times d}$, $W^i \in \mathbb{R}^{m\times m}$ for $i = 2, 3, 4$, $W^5 \in \mathbb{R}^{1\times m}$, $b^j \in \mathbb{R}^m$ for $j = 1, \ldots, 4$ and $b^5 \in \mathbb{R}$. With a slight abuse of notation, the application of the activation function is understood elementwise: $\sigma(z)$ is the vector $(\sigma(z_1), \ldots, \sigma(z_m))$ for $z = (z_1, \ldots, z_m) \in \mathbb{R}^m$. By default, we set $\mu = 0.03$ in the definition of σ.

Also the network $\hat{v}_h$ has an architecture of this type; however, generally with a different parameter m. As indicated in the previous section, we denote the weights of $\hat{v}_h$ by η.

Adam Optimizer, Loss Functions and Algorithm. We implemented the neural networks with the help of the Tensorflow Sequential API. Both neural networks were trained with the Adam Optimizer, a variant of the stochastic gradient descent method based on an adaptive estimation of first-order and second-order moments that improves the speed of convergence [13]. For the optimization parameters, we use the standard values $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$ in the notation of [13].

There are three distinct optimization scenarios with their respective loss functions:

(1) The approximation of u_0 by $u_h^0 \in \mathbb{U}_h$ in line 1 of Algorithm 1: It is a supervised learning problem with the loss function $\mathcal{L} = \|u_0 - w_h\|_{L^2}^2$. We use the constant learning rate $\alpha = 10^{-3}$.

(2) The maximization of $\widetilde{\phi}_h^*$ in line 6 of Algorithm 1: We use the constant learning rate $\alpha = 10^{-5}$. By default the training extends over 500 epochs.

(3) The minimization of $\widetilde{\Phi}_n$ in line 7 of Algorithm 1: We employ the k-dependent decaying learning rate

$$\begin{aligned}\alpha(k) = {} & 10^{-5}\mathbf{1}_{\{k\leq 5\}} + 10^{-6}\mathbf{1}_{\{5<k\leq 50\}} + 10^{-7}\mathbf{1}_{\{50<k\leq 120\}} \\ & + 10^{-8}\mathbf{1}_{\{120<k\leq 140\}} + 10^{-9}\mathbf{1}_{\{140<k\leq 180\}} + 10^{-10}\mathbf{1}_{\{180<k\}}.\end{aligned}$$

By default the training extends over 50 epochs.

The integrals appearing in these loss functions are evaluated with the help of a Monte-Carlo integration method, using the sampling points $\{x_i : i = 1, \ldots, N_s\} \subset \overline{\Omega}$. Here $N_s = N_i + N_b$, with N_i points drawn from a uniform distribution in Ω and N_b points drawn from a uniform distribution on $\partial\Omega$.

Finally, the default termination criterion in line 4 of Algorithm 1 is

$$\text{termination\_criterion}(u_h^{n,k}, k) = \begin{cases} \text{TRUE} & : \ k > 200\,, \\ \text{FALSE} & : \ k \leq 200\,, \end{cases}$$

which is employed in all numerical experiments of the forthcoming section.

5. Numerical results

We consider problem (2.5), with $f = 0$, on the domain $\Omega = (0, \pi)^d$, for $d = 2, 3, 5, 7$. Given the initial condition $u_0(x) = \prod_{i=1}^d \sin(a_i x_i)$, for $a \in \mathbb{N}^d$ and $x \in \overline{\Omega}$, the exact solution to (2.7) with $\kappa = [\sum_{i=1}^d a_i^2]^{-1}$ is $u(t, x) = e^{-t} \prod_{i=1}^d \sin(a_i x_i)$.

We investigate the following types of approximation errors:

$$\text{MSE} = \frac{1}{N_s} \sum_{i=1}^{N_s} (u(t_n, x_i) - u_h^n(x_i))^2,$$

$$\varepsilon_{\text{abs}, L^\infty} = \max_{i=1,\ldots,N_s} |u(t_n, x_i) - u_h^n(x_i)|, \ \varepsilon_{\text{rel}, L^2} = \left[\frac{\sum_{i=1}^{N_s} (u(t_n, x_i) - u_h^n(x_i))^2}{\sum_{i=1}^{N_s} (u(t_n, x_i))^2} \right]^{\frac{1}{2}},$$

$$\varepsilon_{\text{rel}, H^1} = \left[\frac{\sum_{i=1}^{N_s} (u(t_n, x_i) - u_h^n(x_i))^2 + |\nabla u(t_n, x_i) - \nabla u_h^n(x_i)|^2}{\sum_{i=1}^{N_s} (u(t_n, x_i))^2 + |\nabla u(t_n, x_i)|^2} \right]^{\frac{1}{2}},$$

where MSE stands for mean square error, and the other quantities define approximations of the L^∞-norm error and of the relative L^2- and H^1-norm errors, respectively.

In addition the Brezis–Ekeland functional itself represents a measure of the accuracy of the deep learning algorithm since we look for u such that $\Phi(u) = \min \Phi = 0$. It follows that values of the loss function $\widetilde{\Phi}_n$ give us information about the quality of the training and the approximate solution u_h^n.

Unless otherwise stated, we use the layer width $m = m_{v_h} = 30$ for the neural networks $\hat{v}_h$, while the the layer width $m = m_{u_h}$ will be varied for $\hat{u}_h$ depending on the dimension d. For the time discretization we use uniform time steps $\Delta t_n = \Delta t$, $n = 1, \ldots, N$, where we always choose $\Delta t = 10^{-4}$. Finally, for the boundary value penalty parameter we always use $\lambda = 100$.

Energy landscapes for a 5D problem. Let $d = 5$ and $a = (2, 2, 1, 2, 3)^\intercal$. We choose $N_i = 10^5$ inner and $N_b = 10^3$ boundary sampling points. The optimization to obtain the initial value approximation u_h^0 is done over $5 \cdot 10^4$ epochs. We use $m_{u_h} = 60$ for $\hat{u}_h$.

We are interested in the shape of the graphs of the two objective functions $\widetilde{\Phi}_n$ and $\widetilde{\phi}_h^*$ as functions of the neural network weights θ and η, respectively. This will allow us to gain insight into how challenging the training of the neural nets is.

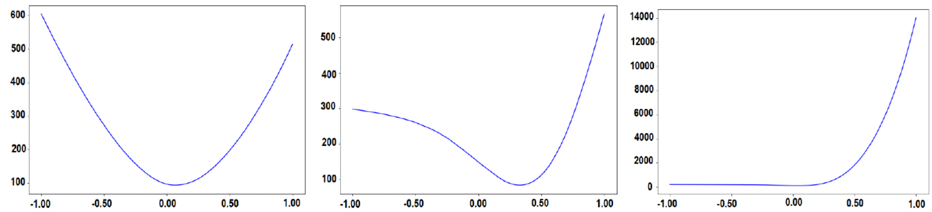


FIG. 1. Plots of $\widetilde{\Phi}_n(\hat{u}_h(\theta); p_h^{1,200})$ against different components of θ: b^1_{60} (left), $W^3_{46,60}$ (middle) and $W^3_{7,45}$ (right).

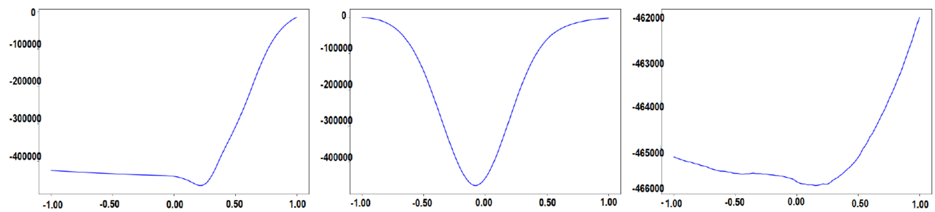


FIG. 2. Plots of $-\widetilde{\phi}_h^*((u_h^{1,199} - u_h^0)/\Delta t; \hat{v}_h(\eta))$ against different components of η: b^2_{30} (left), $W^3_{2,1}$ (middle) and $W^4_{21,3}$ (right).

Having computed $u_h^1 = u_h^{1,K} = \hat{u}_h(\hat{\theta})$, in Figure 1 we plot the loss function $\widetilde{\Phi}_n(\hat{u}_h(\theta); p_h^{n,K})$, for $n = 1$ and $K = 200$, against selected entries of θ. In particular, for each plot we keep all the weights in $\theta = \hat{\theta}$ fixed, apart from a single entry of θ, that we continuously vary from -1 to 1. In this way it is possible to visualize how the Brezis–Ekeland functional varies depending on certain parameters of the neural network $\hat{u}_h$. While generally smooth, we note that the right plot in Figure 1 shows that $\widetilde{\Phi}_n$ has a nearly vanishing gradient when the parameter $W^3_{7,45}$ varies in $[-1, 0]$, which may require attention during the optimization process.

Similarly, in Figure 2 we show the loss function $-\widetilde{\phi}_h^*((u_h^{n,k-1} - u_h^{n-1})/\Delta t; \hat{v}_h(\eta))$ plotted against selected entries of the neural network weights η. Once again we observe nearly flat parts in the graph, but now in addition we see also some non-convex and non-smooth regions, which may pose challenges during the optimization.

Error quantities for a 5D problem. We use the previous example to compute error quantities for the trained neural networks. Table 1 shows values of the Brezis–Ekeland loss function ($\widetilde{\Phi}_n$), the mean square error (MSE), the absolute error ($\varepsilon_{\mathrm{abs},L^\infty}$) and the relative errors ($\varepsilon_{\mathrm{rel},L^2}$, $\varepsilon_{\mathrm{rel},H^1}$) for the neural networks u_h^n for every t_n, $n = 0, \ldots, N$. The reported values of $\widetilde{\Phi}_n$ are large, possibly caused by the fact that the measure of Ω is considerable with $|\Omega| = \pi^5 \approx 306$, as is the scaling of the boundary term $\lambda \| \cdot \|^2_{L^2(\partial\Omega)}$ with $\lambda = 100$ and $|\partial\Omega| = 2 \cdot 5 \cdot \pi^4 \approx 974$. We observe that all the other error quantities slowly increase with time, which is typical for approximations of parabolic PDEs.

	$\widetilde{\Phi}_n$	MSE	$\varepsilon_{abs,L^\infty}$	ε_{rel,L^2}	ε_{rel,H^1}
t_0	—	5.894e-04	0.142	0.138	0.420
t_1	112.762	6.195e-04	0.134	0.142	0.438
t_2	67.164	6.516e-04	0.146	0.140	0.439
t_3	72.938	6.785e-04	0.145	0.149	0.439
t_4	62.323	7.214e-04	0.150	0.153	0.441
t_5	51.332	7.466e-04	0.150	0.156	0.442
t_6	85.143	7.804e-04	0.158	0.160	0.444
t_7	45.920	8.211e-04	0.161	0.164	0.445
t_8	48.652	8.196e-04	0.162	0.164	0.446
t_9	58.798	8.686e-04	0.167	0.169	0.447
t_{10}	37.395	8.780e-04	0.166	0.170	0.448

TAB. 1. Error quantities at times t_n, $n = 0, \dots, N$, for the 5D test problem.

Apart from the global error properties, we are also interested in how these quantities change during the training process. In Figure 3 we plot the loss function $\widetilde{\Phi}_n(u_h^{n,k}; p_h^{n,k})$ and the four contributions to it against k, for $k = 1, \dots, K = 200$, during the training for the time $t_4 = 4 \cdot 10^{-4}$, i.e. $n = 4$. We observe a significant decrease of the Brezis–Ekeland functional $\widetilde{\Phi}_n$ during the training, from about 400 for $k = 1$ to about 60 for $k = 200$, when the weights seem to have converged. Observe also that the decrease is non-monotone, with a global maximum of about 1000, and that the graph is rather oscillatory. In addition, we note that after an initial increase, the functional decays rapidly at first and then slower as the iteration proceeds. The plot of the four contributions reveals that the term $\lambda \|u_h^{n,k}\|^2_{L^2(\partial\Omega)}$ is the dominant contribution in the Brezis–Ekeland functional (3.3) once the iterative scheme settles down.

The analogous plot for the mean square error (MSE) is shown on the left of Figure 4, where we again notice an oscillatory decrease until convergence is reached. In addition, on the right of Figure 4 we show the concatenated plots of the MSE against k, for every time t_n, $n = 1, \dots, N$. To help differentiate the different time steps, we indicate the start of the training for a new time step with vertical lines. The figure demonstrates that overall the MSE increases in time, but that each training procedure decreases the MSE until convergence can be observed.

Dependence on the dimension d. Here we let $d = 2$, 3 or 5, and set $a = (2, 2)^\intercal$, $a = (2, 2, 3)^\intercal$ and $a = (2, 2, 1, 2, 3)^\intercal$, respectively. The number of sampling points in Ω is $N_i = 10^4$ for $d = 2$ and $N_i = 10^5$ for $d = 3, 5$, while $N_b = 400, 600, 1000$ for $d = 2, 3, 5$, respectively. Moreover, for the training of the initial conditions u_h^0 we use $5 \cdot 10^3$ epochs in the cases $d = 2, 3$, and $5 \cdot 10^4$ for $d = 5$. We set $m_{u_h} = 60$ throughout.

Table 2 shows the values of the Brezis–Ekeland loss function $\widetilde{\Phi}_n$, the MSE, the absolute error $\varepsilon_{\mathrm{abs},L^\infty}$, the relative error $\varepsilon_{\mathrm{rel},L^2}$ and the GPU time used for the training of the neural networks $\hat{u}_h$ and $\hat{v}_h$, for the tenth time step, $t_{10} = 0.001$, for

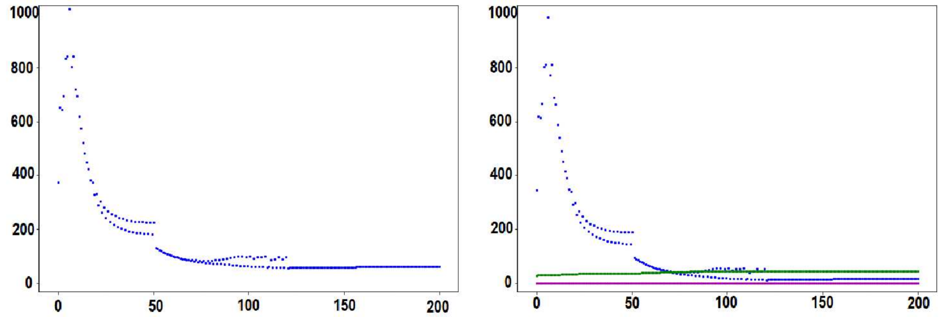


FIG. 3. Plots of $\widetilde{\Phi}_n(u_h^{n,k}; p_h^{n,k})$ (left) and of the 4 terms contributing to it (right) against k, for $n = 4$, for the 5D test problem. For the right plot the chosen colours are red for $\frac{\kappa \Delta t}{2} \|\nabla u_h^{n,k}\|_{L^2}^2$, blue for $\Delta t\, p_h^{n,k}$, magenta for $(u_h^{n,k} - u_h^{n-1}, u_h^{n,k})$ and green for $\lambda \|u_h^{n,k}\|_{L^2(\partial\Omega)}^2$.

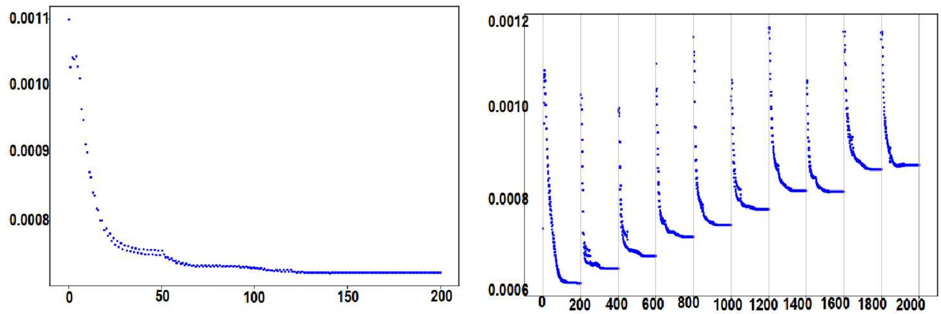


FIG. 4. Plots of the MSE against k at time $t_4 = 4 \cdot 10^{-4}$ (left), and of the MSE against $(n-1)K + k$, for $n = 1, \ldots, N$. (right)

the three problems $d = 2, 3, 5$. It can be observed that an increase in the dimension leads to a decrease in the accuracy of the algorithm. Indeed, reading the table from left to right, we notice that all the measures increase monotonically with the dimension, with the only exception being the absolute error $\varepsilon_{\mathrm{abs},L^\infty}$, which actually reduces slightly going from the 3D to the 5D problem. While a growth of the error is to be expected with an increase of the dimension, the figures here indicate that further improvements of the neural net architecture and the training methodology should be investigated.

Effect of the number of nodes m for a 7D problem. Let $d = 7$ and $a = (2, 2, 1, 3, 2, 2, 3)^\intercal$. We choose $N_i = 10^5$ and $N_b = 1400$ sample points. The initial conditions are training over $5 \cdot 10^4$ epochs. The layer width m_{u_h} of the networks $\hat{u}_h$ varies between 60 and 100.

	2D	3D	5D
$\widetilde{\Phi}_n$	4.54e-02	3.97	39.93
MSE	1.74e-04	1.00e-03	7.67e-03
$\varepsilon_{abs,L^\infty}$	6.25e-03	0.14	0.13
ε_{rel,L^2}	2.71e-02	8.98e-02	0.15
GPU time [s]	58195	73595	100641

TAB. 2. Comparison of error quantities and the GPU time for the three test problems with $d = 2, 3, 5$.

m_{u_h}	60	100
$\widetilde{\Phi}_n$	469.11	60.44
MSE	3.80e-03	1.21e-03
$\varepsilon_{abs,L^\infty}$	0.59	0.31
ε_{rel,L^2}	0.70	0.38
GPU time [s]	119972	117657

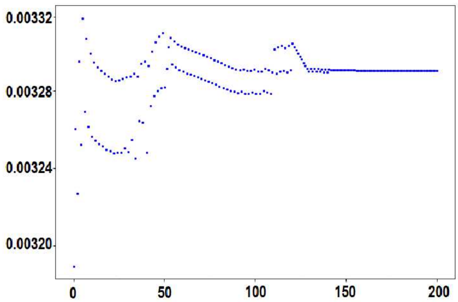
TAB. 3. Comparison of error quantities and the GPU time for the 7D test problem with either 60 or 100 nodes per layer.

From the error quantities reported in Table 3 we can immediately see that using more nodes in the network architecture is beneficial in higher dimensions. In fact, all the reported quantities are lower when considering the case with $m_{u_h} = 100$.

Finally, in Figure 5 we present plots of the MSE against $k = 1, \ldots, K = 200$ for the training of the third time step, $n = 3$, comparing the performance of the two different network structures. We note that the MSE is on average increasing for the network with 60 nodes per layer, possibly indicating that the neural network does not have enough expressivity to make the learning effective. In contrast, when training with wider layers one is able to decrease the MSE after the typical initial increase. These results strongly indicate that for higher dimensional problems, more elaborate networks perform better in practice. Unfortunately, memory and GPU time limitations mean that at present we are not able to investigate this trend for even higher dimensional problems.

6. Conclusions

We introduced a novel deep learning approach for the numerical solution of PDEs using the Brezis–Ekeland principle. As a proof of concept we implemented a practical algorithm for the heat equation and presented results for experiments up to dimension 7. Higher dimensional problems are particularly computationally challenging, and more research into the optimal design for the employed neural networks is needed. Similarly to other neural network based approaches, we expect our method to be superior to classical algorithms for very high dimensional problems, whereas in lower space dimensions classical algorithms are likely more efficient. In



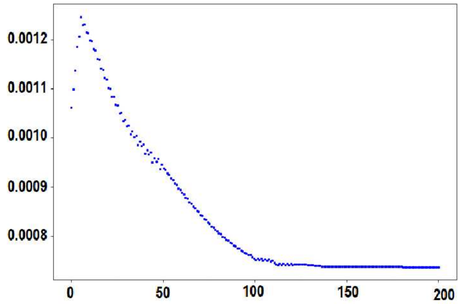


FIG. 5. Plots of the MSE against k at time $t_3 = 3 \cdot 10^{-4}$ for the 7D problem with neural networks using 60 (left) and 100 (right) nodes per layer.

addition, an extension of the implemented method to nonlinear problems is part of future research.

References

[1] Blechschmidt, J., Ernst, O.G., *Three ways to solve partial differential equations with neural network - a review*, GAMM-Mitt. **44** (2021), no. 2, Paper No. e202100006, 29.

[2] Brézis, H., Ekeland, I., *Un principe variationnel associé à certaines équations paraboliques. Le cas dépendant du temps*, C. R. Acad. Sci. Paris Sér. A-B **282** (1976), no. 20, Ai, A1197–A1198.

[3] Brezis, H., Ekeland, I., *Un principe variationnel associé à certaines équations paraboliques. Le cas indépendant du temps*, C. R. Acad. Sci. Paris Sér. A-B **282** (1976), no. 17, Aii, A971–A974.

[4] Carini, L., *BENNO*, https://github.com/LauraCarini/BENNO (2022).

[5] Di Pietro, D.A., Ern, A., *Mathematical aspects of discontinuous Galerkin methods*, Mathématiques & Applications (Berlin), vol. 69, Springer, Heidelberg, 2012.

[6] E, W., Yu, B., *The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat. **6** (2018), no. 1, 1–12.

[7] Ghoussoub, N., *Self-dual partial differential systems and their variational principles*, Springer Monographs in Mathematics, Springer, New York, 2009.

[8] Han, J., E, W., *Deep Learning Approximation for Stochastic Control Problems*, arXiv **cs.LG** (2016).

[9] Han, J., Jentzen, A., E, W., *Solving high-dimensional partial differential equations using deep learning*, Proceedings of the National Academy of Sciences of the United States of America **115** (2018), no. 34, 8505–8510.

[10] Henry-Labordère, P., *Deep Primal-Dual Algorithm for BSDEs: Applications of Machine Learning to CVA and IM*, SSRN Electronic Journal (2017).

[11] Kaltenbach, A., Zeinhofer, M., *The Deep Ritz Method for Parametric p-Dirichlet Problems*, arXiv:2207.01894, 2022.

[12] Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., Yang, L., *Physics-informed machine learning*, Nature Reviews Physics **3** (2021), no. 6, 422–440.

[13] Kingma, D.P., Ba, J., *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980, 2014.

[14] Raissi, M., *Forward-Backward Stochastic Neural Networks: Deep Learning of High-dimensional Partial Differential Equations*, arXiv **stat.ML** (2018).

[15] Raissi, M., Karniadakis, G.E., *Hidden physics models: machine learning of nonlinear partial differential equations*, J. Comput. Phys. **357** (2018), 125–141. DOI: 10.1016/j.jcp.2017.11.039

[16] Raissi, M., Perdikaris, P., Karniadakis, G.E., *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. DOI: 10.1016/j.jcp.2018.10.045

[17] Roubíček, T., *Nonlinear partial differential equations with applications*, second ed., International Series of Numerical Mathematics, vol. 153, Birkhäuser/Springer Basel AG, Basel, 2013.

[18] Sirignano, J., Spiliopoulos, K., *DGM: a deep learning algorithm for solving partial differential equations*, J. Comput. Phys. **375** (2018), 1339–1364. DOI: 10.1016/j.jcp.2018.08.029

[19] Stefanelli, U., *The Brezis-Ekeland principle for doubly nonlinear equations*, SIAM J. Control Optim. **47** (2008), no. 3, 1615–1642.

[20] Stefanelli, U., *The discrete Brezis-Ekeland principle*, J. Convex Anal. **16** (2009), no. 1, 71–87.

[21] Visintin, A., *Extension of the Brezis–Ekeland–Nayroles principle to monotone operators*, Adv. Math. Sci. Appl. **18** (2008), no. 2, 633–650.

Dipartimento di Mathematica, Università di Trento,
38123 Trento, Italy
E-mail: `laura.carini@studenti.unitn.it`

Mathematics Department, University College London,
25 Gordon Street, London, WC1H 0AY, United Kingdom
E-mail: `max.jensen@ucl.ac.uk`

Dipartimento di Mathematica, Università di Trento,
38123 Trento, Italy
E-mail: `robert.nurnberg@unitn.it`