Communications in Mathematics and Statistics (2023) 11:21–57
https://doi.org/10.1007/s40304-023-00339-5

Neural Network-Based Variational Methods for Solving Quadratic Porous Medium Equations in High Dimensions

Min Wang[1] · **Jianfeng Lu**[1]

Received: 14 May 2022 / Revised: 17 October 2022 / Accepted: 2 February 2023 /
Published online: 9 March 2023

Abstract

In this paper, we propose and study neural network-based methods for solutions of high-dimensional quadratic porous medium equation (QPME). Three variational formulations of this nonlinear PDE are presented: a strong formulation and two weak formulations. For the strong formulation, the solution is directly parameterized with a neural network and optimized by minimizing the PDE residual. It can be proved that the convergence of the optimization problem guarantees the convergence of the approximate solution in the L^1 sense. The weak formulations are derived following (Brenier in Examples of hidden convexity in nonlinear PDEs, 2020) which characterizes the very weak solutions of QPME. Specifically speaking, the solutions are represented with intermediate functions who are parameterized with neural networks and are trained to optimize the weak formulations. Extensive numerical tests are further carried out to investigate the pros and cons of each formulation in low and high dimensions. This is an initial exploration made along the line of solving high-dimensional nonlinear PDEs with neural network-based methods, which we hope can provide some useful experience for future investigations.

Keywords Quadratic porous medium equation · High-dimensional nonlinear PDE, Neural network · Variational formulation · Deep learning

Mathematics Subject Classification 68T07 · 65M12 · 35A15

✉ Min Wang
wangmin@math.duke.edu

[1] Duke University, Durham, NC, USA

1 Introduction

Solving high-dimensional PDEs is a long-standing challenge in scientific computing. Standard mesh-based methods, such as Finite Element Method (FEM), Finite Difference Method (FDM) would suffer from the curse of dimensionality, i.e., in order to sustain the accuracy of the approximate solution to high-dimensional PDEs, an approximation space with exponentially large size must be used. The number of degrees of freedom associated to the approximation space is often proportional to the number of elements in the mesh which usually scales exponentially in the dimension to achieve a suitable discretization of the domain. Therefore, for high-dimensional problems, the mesh-based methods are impractical. Alternatively, semilinear parabolic PDEs can also be solved pointwisely based on stochastic representation of the solutions using Monte Carlo algorithm [12, 13, 27, 28], but such approaches only apply to specific type of PDEs.

To circumvent the challenges for solving general high-dimensional (nonlinear) PDEs, many attempts have been made. One natural idea is to restrict to a solution ansatz. For example, using the tensor train (TT) format to approximate the solutions of high-dimensional PDEs [4, 6–9, 23]. While such methods are quite successful if the solution can be well represented by the tensor train, the representability is not guaranteed. Another natural and promising candidate for PDE solution ansatz is the artificial neural networks. Thanks to the rich expressiveness of the neural networks to parametrize high dimensional functions [2]. Theoretical results are also available to justify the approximability of PDE solutions by neural networks without curse of dimensionality, e.g., [14, 18]. Many recent works have been devoted for various approaches of using neural networks to solve high-dimensional PDEs. Typically, such methods first identify functional optimization problems corresponding to the PDEs

$$\underbrace{u}_{\text{PDE solution}} = \underbrace{\arg\min_f \mathcal{C}(f)}_{\text{PDE-inspired optimization problem}} .$$

Then one could take neural network as an ansatz of the minimizer u and minimize the parameters using stochastic gradient type approaches.

One well known method of this kind is the physics informed neural network (PINN) [22], which takes the loss function to be directly the PDE residual. One drawback of PINN is that, to compute the loss function, the derivatives or high-order derivatives of the neural network need to be computed throughout the training process. However, the generality and simplicity of this framework still make it an easy choice when it comes to the high-dimensional PDEs. Other neural network-based methods to solve PDEs include the Deep Ritz Methods [3], the Deep BSDE method [10, 11] for semilinear parabolic PDEs, utilizing different formulations to turn the high dimensional PDE problems into an optimization problem on the neural network parameters. More recent work [30] adopts weak variational formulation for linear equations to solve such PDEs using neural networks. While it is generically unclear how to extend such techniques to nonlinear ones.

In this paper, we make an attempt along the direction of utilizing weak formulation for solving nonlinear PDEs in high dimension. In particular, we consider high-dimensional quadratic porous medium equation (QPME). The equation will be introduced in Sect. 2, together with a brief review of conventional numerical methods for solving such equations. Several variational formulations of the QPME is proposed to solve such PDE (Sect. 3) which both allow solutions in a very weak sense and can be sought with deep learning techniques. In addition, these formulations are further compared with the PINN formulation. In Sect. 4, more detailed treatment of the neural network for solving the optimization problem is presented. Numerical results are then provided to verify and compare the effectiveness of the proposed methods in Sects. 5 and 6.

2 Preliminaries

We consider the porous medium equation (PME)

$$\partial_t u = \frac{1}{m}\Delta u^m, \quad (t, \mathbf{x}) \in Q.$$

PME is a degenerate parabolic equation. It is only parabolic where $u > 0$. When m is taken to be 2, the quadratic porous medium equation (QPME or Boussinesq's equation) reads

$$\partial_t u = \frac{1}{2}\Delta u^2 = \operatorname{div}(u\nabla u) = \frac{1}{2}u\Delta u + \frac{1}{2}|\nabla u|^2, \quad (t, \mathbf{x}) \in Q, \tag{2.1}$$

where $Q := [0, T] \times \Omega$ and Ω stands for a bounded domain in $\mathbb{R}^d$. $\Delta = \Delta_x$ represents the Laplace operator acting on the space variables. The equation is mainly used to describe process involving fluid flow, heat transfer or diffusion. Particularly, it can be used to study the flow of an isentropic gas through a porous medium [17, 20]. In this case, $u(t, \mathbf{x}) \in \mathbb{R}$ is a scalar function denoting the density of the gas (u^2 is roughly the pressure, and $u\nabla u$ stands for the flux). Physical constraints may apply such that $u > 0$. Power 2 here relates to the thermal dynamics character expansion in terms of the pressure of the gas of interest (linear dependency).

A main feature of PME is the "finite propagation" speed, in contrast with the infinite propagation of usual diffusion equations. Essentially, the free boundary that separates the regions where the solution is positive (i.e., where "there is gas", according to the standard interpretation of u as a gas density), from the "empty region" where $u = 0$ moves as time passes.

$$\Gamma = \partial\mathcal{P}_u \cap Q,$$

where $\mathcal{P}_u := \{(t, \mathbf{x}) \in Q \mid u(t, \mathbf{x}) > 0\}$ denotes the set where u is positive. Γ is also sometimes referred as the moving boundary, propagation fronts or the interface. While a rather comprehensive theoretical analysis of this PDEs is provided in [26], exact

solutions to general initial/boundary conditions can usually not be obtained. Numerical schemes thus must be applied to obtain approximate solutions. Most previous studies of PME from a numerical aspect put their focus on dealing with the moving free boundaries of the solutions. The adaptive moving mesh schemes were proposed and coupled with mesh-based methods such as finite element method (FEM) to obtain accurate yet efficient numerical solutions to PME [21]. However, such methods can not be used to solve high-dimensional QPME due to the curse of dimensionality. The only exception as far as we know is [25], in which the supervised learning was conducted to learn the correspondence between the physical parameters in the PDE and the one-dimensional solutions at certain evaluation x, the learning of the global solution is not considered by the authors.

While PME is mainly used in modelings for low physical dimensions, $d = 2$ or 3. In this work, we use it as a prototypical degenerate nonlinear equation in high dimensions to test numerical PDE solvers based on the neural networks. The high-dimensional diffusion might be used for certain machine learning tasks such as analysis of high dimensional point cloud data, which we will leave for future investigations.

3 Variational formulations of QPME

Since the mesh-based algorithms suffer from the curse of dimensionality, we therefore turn to neural network-based techniques for solutions to high-dimensional PDEs. In particular, we first convert the initial/boundary value problem (IBVP) of QPME into a variational formulation and then take the neural network as an ansatz of the solution. The objective function is then taken as the loss function and the extrema will be obtained by optimizing the loss function with stochastic gradient descent (SGD) or its variants.

In this section, we specifically focus on the first step of this procedure, i.e., the IBVP and the variational reformulation.

3.1 Initial/boundary value problem

Consider the QPME on a hyperrectangle

$$\partial_t u = \frac{1}{2}\Delta u^2, \quad (t, \mathbf{x}) \in Q,$$

where $Q = [0, T] \times \Omega$ and $\Omega = [-a_i, a_i]^d$. We consider the QPME with the homogeneous Dirichlet boundary condition

$$\textbf{Dirichlet B.C.} \quad u(t, \mathbf{x})|_{\Sigma_T} = 0, \tag{3.1}$$

where $\Sigma_T := [0, T] \times \partial\Omega$. We also impose the initial condition to the PDE as

$$\textbf{I.C.} \quad u(0, \mathbf{x}) = u_0(\mathbf{x}) \quad \mathbf{x} \in \Omega. \tag{3.2}$$

3.2 Strong formulation

One immediate optimization formulation is to use the strong form of the PDE by minimizing the squared PDE residual

$$\mathcal{L}_{\text{PDE}}(u) = \int_Q \left(\partial_t u - \frac{1}{2}\Delta u^2\right)^2 d\mathbf{x}dt. \tag{3.3}$$

If both the I.C. and B.C. are strictly enforced as hard constraints, the optimization problem can then be formulated as

$$\min_{u \in V_0} \mathcal{L}_{\text{PDE}}(u), \tag{3.4}$$

where $V_0 := \{f : f|_{\Sigma_t} = 0, f(0, \mathbf{x}) = u_0(\mathbf{x})\}$. Alternatively, both I.C. and B.C. can be treated as soft constraints enforced by penalizations: We may define

$$\mathcal{L}_B(u) = \int_{\Sigma_T} u^2 \, d\mathbf{x}dt \tag{3.5}$$

for homogeneous Dirichlet boundary condition and

$$\mathcal{L}_I(u) := \int_\Omega \left(u\left(0, \mathbf{x}\right) - u_0\left(\mathbf{x}\right)\right)^2 \, d\mathbf{x} \tag{3.6}$$

for the initial condition. The optimization problem (3.4) can then be relaxed to

$$\min_{u \in V} \mathcal{L}_{\text{PINN}}(u) \tag{3.7}$$

for some function space V, where

$$\mathcal{L}_{\text{PINN}}(u) := \kappa \mathcal{L}_{\text{PDE}}(u) + \mu \mathcal{L}_B(u) + \nu \mathcal{L}_I(u) \tag{3.8}$$

is weighted sum of the PDE residual, the error of boundary condition and the error for initial condition. κ, μ, ν are weights for each term. We used the subscript PINN for the loss function, as such formulation was popularized by the PINN method [22] in recent years, while the idea dates back to early days of using neural network ansatz for PDE solutions [16].

So far, the PDE residual, mismatch in initial condition and boundary condition of u are all measured in an L^2 sense. We could also define an analogous L^1 optimization

problem (3.7) with:

$$\begin{aligned}\mathcal{L}_{\mathrm{PDE}}(u) &= \int_Q \left|\partial_t u - \frac{1}{2}\Delta u^2\right| \, d\mathbf{x}dt,\\ \mathcal{L}_B(u) &= \int_{\Sigma_T} |u| \, d\mathbf{x}dt,\\ \mathcal{L}_I(u;u_0) &:= \int_\Omega |u(0,\mathbf{x}) - u_0(\mathbf{x})| \, d\mathbf{x}.\end{aligned} \tag{3.9}$$

We refer the target function in L^1 by $\mathcal{L}_{\mathrm{PINN}-L^1}$ and that in L^2 by $\mathcal{L}_{\mathrm{PINN}-L^2}$. While using L^2 to measure the PDE residual and I.C./B.C. mismatch is a standard practice in PINN, the use of L^1 is inspired by the following stability analysis.

Assume u is a strong solution to the homogeneous QPME (2.1) subject to the homogeneous Dirichlet boundary condition and the initial condition (3.2). Let $\hat{u}(t,\mathbf{x})$ be another smooth function that satisfies the Dirichlet boundary condition $\hat{u}(t,\mathbf{x})|_{\Sigma_t} = 0$. We define $\hat{f}(t,\mathbf{x}) := \partial_t\hat{u}(t,\mathbf{x}) - \frac{1}{2}\Delta\hat{u}^2(t,\mathbf{x})$, so that $\hat{u}$ satisfies the QPME with $\hat{f}$ being the forcing term

$$\begin{aligned}&\partial_t\hat{u} = \frac{1}{2}\Delta\hat{u}^2 + \hat{f}, \quad \forall(t,\mathbf{x}) \in Q,\\ &\hat{u}(0,\mathbf{x};\theta) = \hat{u}_0(\mathbf{x}), \quad \forall\mathbf{x} \in \Omega,\end{aligned}$$

where $\hat{u}_0(\mathbf{x}) = \hat{u}(0,\mathbf{x})$.

Then the following L^1 contraction holds for any $t \in [0,T]$ [26]:

$$||u(t) - \hat{u}(t)||_1 \le ||u_0 - \hat{u}_0||_1 + \int_0^t ||f(s) - \hat{f}(s)||_1 \, ds. \tag{3.10}$$

Thus,

$$||u(t) - \hat{u}(t)||_1 \le ||u_0 - \hat{u}_0||_1 + \int_0^t ||\partial_s\hat{u} - \frac{1}{2}\Delta\hat{u}^2||_1 \, ds \le \tilde{C}\mathcal{L}_{\mathrm{PINN}-L^1}(\hat{u}) \tag{3.11}$$

noticing $f(t)$ in (2.1) is zero. Therefore, as long as Ω is a bounded domain, by Cauchy–Schwarz inequality, we have

$$||u(t)-\hat{u}(t)||_1^2 \le C\Big(||u_0-\hat{u}_0||_2^2 + \int_0^t ||\partial_s\hat{u} - \frac{1}{2}\Delta\hat{u}^2||_2^2 \, ds\Big) \le \tilde{C}\mathcal{L}_{\mathrm{PINN}-L^2}(\hat{u}). \tag{3.12}$$

The estimates (3.11) and (3.12) show that if the QPME is solved by minimizing the loss $\mathcal{L}_{\mathrm{PINN}}$, the convergence of the optimization guarantees the convergence to the exact solution to (2.1) in L^1 sense. In other words, if the problem is further solved with deep learning methods, the loss is essentially a direct real time indicator of the approximation error during the entire training process. It can thus be naturally used to truncate the training when needed. Additionally, this result suggests that the intrinsic norm to measure the approximation error is L^1 norm.

3.3 Relaxed concave optimization problem

Besides the strong formulation, in this section, we derive and consider a series of optimization problems which can also be used to solve the QPME; which correspond to various weak formulation of the PDE.

We start by considering the very weak solutions to the QPME (2.1), i.e., $u \in L^1(Q)$ satisfying

$$\int_Q \left(-2\partial_t \psi u - \Delta \psi u^2 + 2u_0 \partial_t \psi\right) d\mathbf{x}dt = 0 \tag{3.13}$$

for all test functions $\psi \in C^{2,1}(\bar{Q})$ which vanishes on Σ_T and for $t = T$. Essentially, a very weak solution is an integrable distribution solution. Unlike strong solutions, no derivative of the solution is used in defining such solutions; so very weak solutions have much lower regularity requirements. We also remark that while we focus on very weak solutions in this paper, there are different ways of defining generalized solutions for QPME. A weak solution, for example, is defined to be a function u such that $u^2 \in L^1_{loc}(0, T;\ W^{1,1}_{loc})$ which satisfies

$$\int_Q \left(-2\partial_t \psi u + \nabla(u^2) \cdot \nabla \psi + 2u_0 \partial_t \psi\right) d\mathbf{x}dt = 0.$$

It is clear that all weak solutions are very weak solutions by definition; weak solutions require higher regularity of the solutions.

The following theorem gives a characterization for very weak solutions to QPME [5].

Theorem 3.1 ([5]) *Any very weak solution u to QPME can be recovered as*

$$u = \frac{\partial_t \phi^*}{1 - \Delta \phi^*}, \tag{3.14}$$

where

$$\phi^* = \arg\max_{\phi \in B} J(u_0) = \arg\max_{\phi \in B} \int_Q \left(\frac{-(\partial_t \phi)^2}{1 - \Delta \phi} + 2u_0 \partial_t \phi\right) d\mathbf{x}dt \tag{3.15}$$

with $B := \{\phi \mid \phi(T, \mathbf{x}) = 0,\ 1 - \Delta\phi \geq 0\}$. In addition, any solution ϕ^* satisfies $1 - \Delta\phi^* \geq (\frac{t}{T})^{\frac{d}{d+2}}$.

While we will not repeat the proof here, let us mention that the proof starts with minimizing the Lyapunov ("entropy") functional among very weak solutions u of QPME

$$\int_Q u^2(t, \mathbf{x})\, d\mathbf{x}dt, \tag{3.16}$$

it can then be proved that the following formulations are equivalent letting

- $A := \{u \in L^2(Q)$ is a very weak non-negative solution associated with $u_0 \in L^2(\Omega)\}$,

- $B := \{\phi \mid \phi(T, \mathbf{x}) = 0,\ 1 - \Delta\phi \geq 0\}$.

1. Original form

$$I(u_0) = \inf_{u\in A} \sup_{\phi\in B} \int_Q \left(u^2 - 2\partial_t\phi u - \Delta\phi u^2 + 2u_0\partial_t\phi\right) d\mathbf{x}dt. \tag{3.17}$$

2. Flipping sup, inf

$$J(u_0) = \sup_{\phi\in B} \inf_{u\in A} \int_Q \left(u^2 - 2\partial_t\phi u - \Delta\phi u^2 + 2u_0\partial_t\phi\right) d\mathbf{x}dt. \tag{3.18}$$

3. Pointwise minimization of (3.18)

$$\tilde{J}(u_0) = \sup_{\phi\in B} \int_Q \left(\frac{-(\partial_t\phi)^2}{1 - \Delta\phi} + 2u_0\partial_t\phi\right) d\mathbf{x}dt. \tag{3.19}$$

4. Let $q = \partial_t\phi, \sigma = 1 - \Delta\phi$ in (3.19)

$$\begin{aligned} &\hat{J}(u_0) = \sup_{q,\sigma} \int_Q \left(\frac{-q^2}{\sigma} + 2u_0 q\right) d\mathbf{x}dt, \\ &\sigma \geq 0, \quad \sigma(T, \cdot) = 1, \quad \partial_t\sigma + \Delta\phi = 0. \end{aligned} \tag{3.20}$$

More specifically, it is proved that

$$\int_Q u^2(t, \mathbf{x})\, d\mathbf{x}\, dt = I(u_0) = J(u_0) = \tilde{J}(u_0) = \hat{J}(u_0). \tag{3.21}$$

Theorem 3.1 shows that we can indirectly obtain very weak solutions to QPME by solving (3.15). We first obtain ϕ^*, then obtain candidates for the very weak solution with (3.14). We can therefore consider the following loss function

$$\textbf{ϕ formulation} \quad \mathcal{L}_\phi(\phi) := -\int_Q \left(\frac{-(\partial_t\phi)^2}{1 - \Delta\phi} + 2u_0\partial_t\phi\right) d\mathbf{x}dt. \tag{3.22}$$

It is not hard to see that if a smooth ϕ^* is a minimizer, as long as the recovered solution satisfies the homogeneous boundary condition and the initial condition, $\frac{\partial_t\phi^*}{1-\Delta\phi^*}$ must be a solution to QPME. Moreover, in the case where $u \geq 0$, it has been proved that the solution to the QPME subject to $u_0 \geq 0$ is unique. However, it is worth noting that such minimizer ϕ^* is not necessarily unique. Thus (3.22) can be used to identify the unique solution to QPME, as long as the initial/boundary conditions are imposed, but more than one minimizer ϕ^* could theoretically exist [5].

Moreover, since (3.20) is equivalent to (3.19), one can also recover a candidate of very weak solution to QPME with q^* and σ^* by

$$u^*_{q,\sigma} := \frac{q^*}{\sigma^*} \tag{3.23}$$

with q^* and σ^* being the maximizer of (3.20). Thus, we may also consider the loss function

$$\mathbf{q-\sigma\,formulation}\quad \mathcal{L}_{q,\sigma}(q,\sigma) = -\int_Q \left(\frac{-q^2}{\sigma} + 2u_0 q\right) d\mathbf{x}dt. \tag{3.24}$$

Similar to the discussion in Sect. 3.2, we can also relax the initial/boundary conditions of the recovered solution u obtained from ϕ formulation and q, σ formulation to a penalization by regularizing $\mathcal{L}_B$ and $\mathcal{L}_I$ as defined earlier:

$$\mathcal{L}_{\phi-\mathrm{NN}}(u) := \kappa\mathcal{L}_\phi(u) + \mu\mathcal{L}_B(u) + \nu\mathcal{L}_I(u) \tag{3.25}$$

and

$$\mathcal{L}_{q,\sigma-\mathrm{NN}}(u) := \kappa\mathcal{L}_{q,\sigma}(u) + \mu\mathcal{L}_B(u) + \nu\mathcal{L}_I(u). \tag{3.26}$$

However, it is worth pointing out that it is very difficult to impose initial condition as a hard constraint to solution ansatz for both formulations when the optimization problem is solved with neural networks as only intermediate functions ϕ, q, σ will be parametrized. The boundary conditions, on the other hand, can be explicitly imposed by modifying the solution ansatz.

For $q-\sigma$ formulation in particular, we should also note that, the consistency between q and σ

$$\partial_t\sigma + \Delta q = 0 \tag{3.27}$$

needs to be imposed since they are essential derivatives of the same function ϕ. This condition can be imposed by minimizing the residual of Eq. (3.27)

$$\mathcal{L}_{\partial_t\sigma,\Delta q} = \int_Q (\partial_t\sigma + \Delta q)^2 d\mathbf{x}dt \tag{3.28}$$

or in L^1 sense,

$$\mathcal{L}_{\partial_t\sigma,\Delta q} = \int_Q |\partial_t\sigma + \Delta q|\, d\mathbf{x}dt. \tag{3.29}$$

Thus, $\mathcal{L}_{q,\sigma-\mathrm{NN}}$ can be further modified as

$$\mathcal{L}_{q,\sigma-\mathrm{NN}} = \kappa\mathcal{L}_{q,\sigma}(u) + \mu\mathcal{L}_B(u) + \nu\mathcal{L}_I(u) + \gamma\mathcal{L}_{\partial_t\sigma,\Delta q}. \tag{3.30}$$

Let us remark that the condition (3.27) can also be imposed weakly following the Dirichlet principle, $\mathcal{L}_{\partial_t\sigma,\Delta q}$ can be then replaced by

$$\mathcal{L}_{\partial_t\sigma,\nabla q} := \int_{[0,T]} \left(\frac{1}{2}\int_\Omega |\nabla q|^2\, d\mathbf{x} + \frac{\lambda}{2}\left(\int_\Omega q\, d\mathbf{x}\right)^2 + \int_\Omega \partial_t\sigma q\, d\mathbf{x}\right) dt. \tag{3.31}$$

While related results using $\mathcal{L}_{\partial_t\sigma,\nabla q}$ will not be presented in this paper, we remark that this formulation completely bypasses taking second order derivative of q, which

means less smoothness requirement of the neural network ansatz. However, adding such a term would make the optimization more complicated especially in the high-dimensional cases. Since this term can not be interpreted as a pointwise condition as (3.29), it thus can not benefit much from an efficient sampling scheme (see Sect. 5.1). In addition, the introduction of the extra hyper-parameter λ further increases the difficulty of parameter tuning. From our own experience, the training of the q, σ formulation with the term $\mathcal{L}_{\partial_t \sigma, \nabla q}$ seems extremely challenging if not impossible and is therefore not presented.

4 Solving high-dimensional QPME with neural network ansatz

Neural network is a class of functions that have a certain layered structure, for example the feed-forward fully connected neural network is defined to be

$$\mathcal{NN}(\mathbf{x}; \theta) := W_n g(\cdots g(W_2 g(W_1 \mathbf{x} + b_1) + b_2) \cdots) + b_n. \tag{4.1}$$

In this case, each layer of the network is a composition of a linear transformation and a nonlinear function g acting component-wise. Here, $\theta := [W_1, W_2, \ldots, W_n, b_1, b_2, \ldots, b_n]$ are the trainable parameters.

The idea of neural network-based numerical solver for PDEs is to utilize such a neural network $\mathcal{NN}$ to approximate the function of interest, say u. This is usually achieved by solving an optimization problem

$$u = \arg\min_f \mathcal{C}(f), \tag{4.2}$$

where $\mathcal{C}$ is some suitable objective function. Then one could take a neural network as an ansatz and minimize $\mathcal{C}$ by tuning its parameters θ to get an approximate solution $\mathcal{NN}(\cdot; \theta^*)$ where

$$\theta^* = \arg\min_\theta \mathcal{C}(\mathcal{NN}(\cdot; \theta)). \tag{4.3}$$

The process of optimization is also referred as "training", using the terminology from machine learning. The objective function $\mathcal{C}$ is often referred as the loss function.

In Sect. 3, we have derived a few loss functions which can be used to solve the QPME. In this section, we provide further details on how to solve the aforementioned optimization problems with neural networks, especially on how initial and boundary conditions can be imposed to the neural network as a solution ansatz. In particular, the following conditions are generally considered as a solution ansatz to QPME: 1) the initial condition, 2) the boundary condition, 3) the physical constraint, i.e., $u \geq 0$. In addition, one could consider to impose conditions like $1 - \Delta\phi \geq (\frac{t}{T})^{\frac{d}{d+2}}$ to narrow down the search space as we know by Theorem 3.1 that it is satisfied by the true solution. We will slightly modify existing neural network structure as needed to satisfy the constraints. In this paper, we take the architecture of the neural networks to be feed-forward fully-connected as defined in (4.1), while other architectures could also be considered.

4.1 PINN formulation

To solve QPME with PINN formulation (3.3), we first notice the argmin to (3.7) is exactly a solution to QPME. Then the solution itself can be directly parametrized with a neural network. In particular, to further impose the aforementioned conditions to the solution ansatz, we start with a neural network $\mathcal{NN}_u(t, \mathbf{x}; \theta_u)$ with both time t and spatial coordinates $\mathbf{x}$ as its inputs and denote the collection of trainable parameters as θ_u. Moreover, since we need to compute the PDE residual, which includes the computation of second-order derivative of the solution ansatz, $\mathcal{NN}_u(t, \mathbf{x}; \theta_u)$ must be at least second order differentiable. We thus require activation functions g to be smooth ones such as tanh and softplus functions.

To impose the initial condition (3.2) as a hard constraint, we can parametrize the solution $u(t, \mathbf{x})$ as:

$$u(t, \mathbf{x}; \theta_u) = u_0(\mathbf{x}) + t\mathcal{NN}_u(t, \mathbf{x}; \theta_u).$$

However, in this case, the physical constraint ($u \geq 0$) cannot easily be imposed explicitly. One would hope that by minimizing the PDE residual, the numerical solution obtained would be non-negative.

In the case where the initial condition is imposed softly, the term $\mathcal{L}_I$ defined as in (3.6) will be added as a part of the loss $\mathcal{L}_{\text{PINN}}$ and minimized through training. Meanwhile, the physical constraint of solution can be imposed by parametrization:

$$u(t, \mathbf{x}; \theta_u) = \text{softplus}\left(\mathcal{NN}_u\left(t, \mathbf{x}; \theta_u\right)\right),$$

where the softplus function is given by

$$\text{softplus}(x) = \ln(1 + e^x)$$

which guarantees the solution ansatz to be positive.

As for the boundary condition, the homogeneous Dirichlet boundary condition (3.1) can be imposed as a hard constraint. We take advantage of the function

$$f_{dc}(\mathbf{x}) := \prod_{i=1}^{d} \frac{(a_i - x_i)(a_i + x_i)}{a_i^2} \tag{4.4}$$

so that $f_{dc}(\mathbf{x}) = 0$ for any $\mathbf{x} \in \partial\Omega$. Moreover, we notice that $f_{dc}(\mathbf{0}) = 1$ and $0 \leq f_{dc}(\mathbf{x}) \leq 1$ for all $\mathbf{x} \in \Omega$. The solution ansatz $u(t, \mathbf{x})$ can then be further modified by multiplying f_{dc} to satisfy the boundary condition:

$$\textbf{hard I.C. + hard B.C.} \quad u(t, \mathbf{x}; \theta_u) = u_0(\mathbf{x}) + t f_{dc}(x)\, \mathcal{NN}_u\left(t, \mathbf{x}; \theta_u\right), \tag{4.5}$$

$$\textbf{soft I.C. + hard B.C.} \quad u(t, \mathbf{x}; \theta_u) = f_{dc}(\mathbf{x})\ \text{softplus}\left(\mathcal{NN}_u\left(t, \mathbf{x}; \theta_u\right)\right), \tag{4.6}$$

assuming the homogeneous Dirichlet B.C. is satisfied by u_0.

The benefit of solving PINN is that the convergence of training of $\mathcal{L}_{\text{PINN}}$ guarantees accurate solution, which is justified in (3.11) and (3.12). On the other hand, PINN formulation also has its own limitation that it only allows strong solutions. Solutions with less regularity can not be identified with this formulation.

4.2 ϕ formulation

To solve QPME following the ϕ formulation, we need to parametrize $\phi(t, \mathbf{x})$ in (3.22) instead of the solution u directly. When computing $\mathcal{L}_\phi$ as in (3.22), we also need the ansatz of $\phi(t, \mathbf{x})$ to be at least second-order differentiable. Note that this is a much weaker assumption on the solution ansatz of u compared with the PINN. In particular, no assumption is needed on the smoothness of u directly. We simply take a neural network $\mathcal{NN}_\phi(t, \mathbf{x}; \theta_\phi)$ with smooth activation function as its solution ansatz.

We then note that the minimizers ϕ^* to (3.22) must also satisfy certain conditions in order to obtain reasonable solutions. As suggested by Theorem 3.1, we would like to require ϕ to vanish at $t = T$. We thus let

$$\phi(t, \mathbf{x}; \theta_\phi) = (T - t)\mathcal{NN}_\phi(t, \mathbf{x}; \theta_\phi). \tag{4.7}$$

For the recovered solution u_ϕ

$$u_\phi := \frac{\partial_t \phi}{1 - \Delta\phi}, \tag{4.8}$$

unlike PINN formulations, the solution to QPME is not directly parametrized; thus it is not easy to impose the initial condition as a hard constraint. Instead, we enforce the constraint softly relying on the penalty term $\mathcal{L}_I$. The homogeneous Dirichlet boundary condition, on the other hand, can be softly enforced with the term $\mathcal{L}_B$ or enforced as a hard constraint by modifying the neural network. Essentially, we only need

$$\partial_t \phi|_{\partial\Omega} = 0,$$

which can be achieved using the ansatz

$$\textbf{soft I.C. + hard B.C.} \quad \phi(t, \mathbf{x}; \theta_\phi) = (T - t) f_{dc}(\mathbf{x})\mathcal{NN}_\phi(t, \mathbf{x}; \theta_\phi), \tag{4.9}$$

where $f_{dc}(\mathbf{x})$ is defined as in (4.4). Additionally, while the recovered solution $u_\phi \geq 0$ is desired, this condition cannot be easily imposed by simply modifying the solution ansatz.

Compared with the PINN formulation, using a ϕ-formulation allows solutions in a very weak sense. It potentially can find solutions with less regularity. The smoothness requirement is not directly applied to u_ϕ. However, as described above, a few conditions of ϕ can not be easily enforced. In addition to the positivity of u_ϕ, condition like

$$1 - \Delta\phi \geq \left(\frac{t}{T}\right)^{\frac{d}{d+2}} \tag{4.10}$$

is difficult to enforce as a hard constraint either. While (4.10) is preferable as it can narrow down the search function space for ϕ (since we know the PDE solution would satisfy that), it is not necessary. However, the fact that u_ϕ is not confined to be positive function can potentially cause the training of $\mathcal{L}_{\phi-NN}$ converges to unphysical solutions.

4.3 q - σ formulation

The $q - \sigma$ formulation (3.24) is derived from the ϕ formulation, and thus also inherits a few conditions for ϕ. We first notice that when computing $\mathcal{L}_{q,\sigma}$, no computations of derivatives will be needed. However, when computing $\mathcal{L}_{\Delta q, \partial_t \sigma}$, first-order and second-order derivatives of σ and q are required respectively. We thus can start by parametrizing $q(t, \mathbf{x})$ and $\sigma(t, \mathbf{x})$ with neural networks $\mathcal{NN}_q(t, \mathbf{x}; \theta_q)$ and $\mathcal{NN}_\sigma(t, \mathbf{x}; \theta_\sigma)$ which should be at least first and second order differentiable respectively.

To ensure the positivity of σ, as suggested in (3.20), we further parametrize σ by

$$\sigma(t, \mathbf{x}; \theta_\sigma) = \text{softplus}\left(\mathcal{NN}_\sigma(t, \mathbf{x}; \theta_\sigma)\right).$$

To guarantee that

$$\sigma(T, \cdot) = 1,$$

we modify the above and let

$$\sigma(t, \mathbf{x}; \theta_\sigma) = \text{softplus}\Big(\ln(e-1) + (T-t)\,\mathcal{NN}_\sigma(t, \mathbf{x}; \theta_\sigma)\Big). \tag{4.11}$$

Alternatively, if we also impose the condition (to narrow down the search space)

$$\sigma \geq \left(\frac{t}{T}\right)^{\frac{d}{d+2}}, \tag{4.12}$$

one can also parametrize σ as

$$\sigma(t, \mathbf{x}; \theta_\sigma) = \left(\frac{t}{T}\right)^{\frac{d}{d+2}} + (T-t)\,\text{softplus}\left(\mathcal{NN}(t, \mathbf{x}; \theta_\sigma)\right). \tag{4.13}$$

For the recovered solution

$$u_{q,\sigma} = \frac{q}{\sigma},$$

like ϕ formulation, the initial condition can only be softly imposed. The homogeneous Dirichlet boundary condition can be enforced as a hard constraint, as long as

$$q|_{\partial\Omega} = 0.$$

We thus let

$$q(t, \mathbf{x}; \theta_q) = f_{dc}(\mathbf{x}) \mathcal{NN}_q(t, \mathbf{x}; \theta_q). \tag{4.14}$$

Moreover, to ensure $u_{q,\sigma} \geq 0$, we further let

$$\textbf{soft I.C. + hard B.C.} \quad q(t, \mathbf{x}; \theta_q) = f_{dc}(\mathbf{x}) \text{softplus}\left(\mathcal{NN}_q\left(t, \mathbf{x}; \theta_q\right)\right). \tag{4.15}$$

Similar to ϕ formulation, the $q - \sigma$ formulation allows solutions with less regularity. However, two neural networks will be needed to parametrize a solution to QPME, which could potentially be more challenging to train.

4.4 Empirical loss and training data sampling

To solve the QPME with aforementioned formulations (3.7), (3.25) and (3.30), we need to compute the high-dimensional integrals of the neural network or its derivatives to evaluate the loss functions.

In practice, Monte Carlo methods are usually used to approximate those high-dimensional integrals. The approximate solutions are then obtained by minimizing the surrogate empirical loss functions. Take the PINN formulation as an example, let P_Ω be the uniform probability distributions over the spatial domain Ω and let $\{\mathbf{X}_j\}_{j=1}^n$ to be an independent and identically distributed (i.i.d.) sequence of random variables distributed according to P_Ω. Parallelly, we also define $\mathrm{P}_{[0,T]}$ to be the uniform probability distributions over the spatial domain $[0, T]$ and let $\{T_j\}_{j=1}^n$ be an i.i.d. sequence of random variables distributed according to $\mathrm{P}_{[0,T]}$. Define the empirical loss $\mathcal{L}^n_{\text{PINN}}$ by setting

$$\mathcal{L}^n_{\text{PINN}} = \frac{\kappa}{n} \sum_{j=1}^n \left(\partial_t u(T_j, \mathbf{X}_j) - \frac{1}{2} \Delta u(T_j, \mathbf{X}_j)\right)^2 + \frac{\nu}{n} \sum_{j=1}^n \left(u\left(0, \mathbf{X}_j\right) - u_0\left(\mathbf{X}_j\right)\right)^2 \tag{4.16}$$

for the case where only I.C. is imposed softly and the loss measuring norm is taken to be L^2. Notice that all terms are scaled by $\frac{1}{|\Omega|}$, which does not change the minimizer of the problem but can effectively avoid numerical blowup in evaluating the loss during the training. Similarly, $\mathcal{L}_B$ can also be approximated with points uniformly sampled from $\partial\Omega$ when needed. We further refer such sampled data as training data.

However, a uniform sampling of $\mathbf{X}_j$'s sometimes does not meet the need of our computation especially in the case where the dimension d is very large. Notice that one essential feature of solutions to QPME is that it has a free boundary that separates positive part of the solution from the zeros. In particular, in the case where the solution is a Barenblatt solution, the nonzero values of the solution actually concentrate near the origin. Ideally, one would like to sample points in both non-zero and zero region, to capture the local features of the solution. However, with a fixed budget of training samples, it could happen that all randomly sampled data points only reside in the zero region, which is apparently problematic. In fact, this could become a serious issue for high dimensional problems. For example, when $d = 20$, the probability of sampling the nonzero region of a Barrenblatt solution (5.2) at $t = 2$ within $[-7, 7]^{20}$ can be

computed by the ratio of the volume of the d-ball V_{nonzero} with radius $(22)^{1/2}2^{6/11}$ standing for the non-zero region versus the volume of the hypercubic. It can then be computed that

$$\mathrm{P}_{\text{nonzero}} = \frac{V_{\text{nonzero}}}{14^{20}} \approx 1.57 \times 10^{-8}$$

which means, the non-zero region can rarely be sampled if not impossible. Therefore, we would need an effective sampling scheme which puts larger weights over the non-zero region, so that we can accurately approximate the loss function. Ideally, one could hope for an adaptive important sampling scheme which provides samples of the training data based on a distribution adapted to current status of the parameterized solution and its derivatives throughout the training process. However, especially for high-dimensional problems, such sampling scheme is challenging and computationally expensive to implement. Therefore, a hand-crafted sampling scheme is used instead, which is explained in details in Sect. 5 for specified numerical examples.

5 Numerical example: Barenblatt solution

To test the numerical schemes, we will use a series of special solutions, known as Barenblatt Solutions. They are given by

$$U_m(t, \mathbf{x}; C) := t^{-\alpha}\left(\left(C - \frac{\beta(m-1)}{2}\frac{|\mathbf{x}|^2}{t^{2\beta}}\right)^+\right)^{\frac{1}{m-1}}, \tag{5.1}$$

where $\alpha := \frac{d}{d(m-1)+2}$, $\beta := \frac{\alpha}{d}$, $(s)^+ := \max(s, 0)$ and $C > 0$ is an arbitrary constant. This solution takes a Dirac mass as initial data: $\lim_{t\to 0} u(t, \mathbf{x}) \to M\delta(\mathbf{x})$, where M is a function of the constant C (depends on m and d). In the particular case $m = 2$, the Barenblatt Solution to (2.1) reduces to

$$U_2(t, \mathbf{x}; C) := t^{-\frac{d}{d+2}}\left(C - \frac{1}{2(d+2)}\frac{|\mathbf{x}|^2}{t^{\frac{2}{d+2}}}\right)^+. \tag{5.2}$$

The free boundary $\partial\mathcal{P}_u$ to (5.2) in this case can then characterized by the equation

$$|\mathbf{x}| = r_t$$

with $r_t := \sqrt{2C(2+d)}\, t^{\frac{1}{d+2}}$. We also notice that the solution is scale-invariant:

$$u_\lambda(t, \mathbf{x}) := \lambda^\alpha u(\lambda t, \lambda^\beta \mathbf{x}).$$

The shifted Barenblatt Solution is a strong/weak/very weak solution of PME, which is unique subject to the Dirac initial condition.

Since numerically one cannot set δ function as the initial condition, we specifically consider the following IBVP:

$$\begin{aligned} \partial_t u &= \frac{1}{2}\Delta u^2 \quad (t, \mathbf{x}) \in Q = [0,1] \times \Omega, \\ u(0, \mathbf{x}) &= \left(1 - \frac{1}{2(2+d)}|\mathbf{x}|^2\right)^+. \end{aligned} \tag{5.3}$$

Notice the initial condition is essentially the Barenblatt Solution (5.2) evaluated at $t = 1$ when $C = 1$. The exact solution to (5.3) is therefore the Barenblatt Solution (5.2) with the time shifted:

$$U_2(t, \mathbf{x}) := (t+1)^{-\frac{d}{2+d}} \left(1 - \frac{1}{2(d+2)} \frac{|\mathbf{x}|^2}{(t+1)^{\frac{2}{d+2}}}\right)^+. \tag{5.4}$$

We further let $\Omega = [-a, a]^d$, where a is the smallest integer that is greater than the radius of the free boundary of $U_2(t, \mathbf{x})$ at the terminal time $T = 1$:

$$a := \text{ceil}(r_T),$$

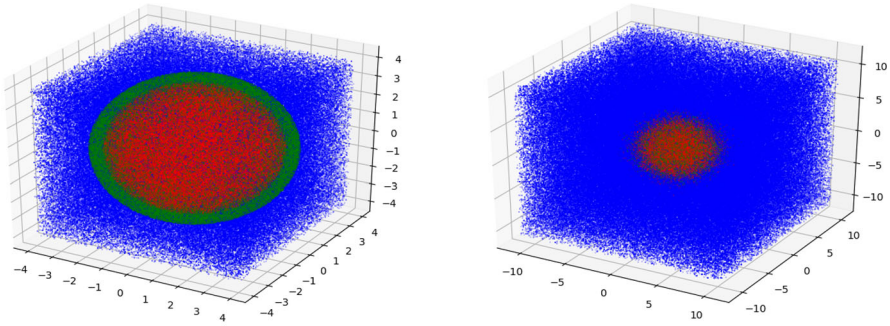
where $r_T = (2+d)^{\frac{1}{2}} 2^{\frac{4+d}{2d+4}}$ to ensure the computational domain is large enough to include the entire free boundary for $t \in [0, 1]$. We take this example to test the effectiveness of the proposed formulations by comparing the approximate solution with the exact one (5.4). The performance of each formulation is further analyzed to show the pros and cons.

5.1 Numerical settings

In particular, We would like to solve the aforementioned QPME with three formulations using neural network ansatz. We specifically take $\mathcal{NN}_u(\cdot, \cdot; \theta_u)$, $\mathcal{NN}_\phi(\cdot, \cdot; \theta_\phi)$, $\mathcal{NN}_q(\cdot, \cdot; \theta_q)$ and $\mathcal{NN}_\sigma(\cdot, \cdot; \theta_\sigma)$ to be fully connected neural networks with two hidden layers and the softplus$(\cdot)$ as their activation function. The corresponding solution ansatz can then be developed following Sect. 4. Notice that when computing the derivatives of the solution ansatz, a chain rule will be applied and the derivatives of $f_{dc}(\mathbf{x})$ need to be computed when a homogeneous Dirichlet B.C. is imposed.

To evaluate the empirical losses, as discussed in Sect. 4.4, we further take randomly sampled data to approximate the integrals in Q or in Ω. When boundary condition is softly imposed, we take additional randomly sampled data over Σ_T to evaluate $\mathcal{L}_B$. In particular, since the data are sampled on the fly, a new set of data will be sampled at each training step, the total number of training data n can then be computed by $n =$ batch size $\times$ training steps.

Moreover, for high-dimensional cases, to make sure the sampled points can capture the features of the solutions, we use the following weighted sampling scheme.



(a) **3d:** $\theta_0 = 0.3, \theta_1 = 0.3$, $\Omega = [-4, 4]^3$. (b) **50d:** $\theta_0 = 0.3, \theta_1 = 0.2$, $\Omega = [-11, 11]^{50}$.

Fig. 1 3D projection (first three coordinates) of samples of $\{\tilde{\mathbf{X}}_j\}_{j=1}^{10^6}$ in Ω. Red: $\tilde{\mathbf{X}}_j \in V_0$, green: $\tilde{\mathbf{X}}_j \in V_1$ and blue: $\tilde{\mathbf{X}}_j \in V_2$. (Colour figure online)

Specifically, we first decompose the region $\Omega = [-a, a]^d$ into $\Omega = V_0 \cup V_1 \cup V_2$, where

$$V_0 := \{\mathbf{x} \in \Omega \mid |\mathbf{x}| \leq r_0\}, \quad V_1 := \{\mathbf{x} \in \Omega \mid r_0 < |\mathbf{x}| \leq r_T\}, \quad V_2 := \{\mathbf{x} \in \Omega \mid |\mathbf{x}| > r_T\}.$$

The radius of these region are decided by the radius of the free boundary to the Barenblat solution (5.4) at $t = 0$ and $t = T = 1$, respectively:

$$r_0 := \sqrt{2(2+d)}, \qquad r_T = (2+d)^{\frac{1}{2}} 2^{\frac{4+d}{2d+4}}.$$

We then take weights θ_0, θ_1 and the $\tilde{\mathbf{X}}_j$'s will be uniformly sampled within V_0 with a probability θ_0, within V_1 with a probability θ_1 and within Ω with a probability $\theta_2 := 1 - \theta_0 - \theta_1$ (see Fig. 1 for an illustration of the sampled training data). The probability of sampling data in each region can thus be computed as

$$P_{V_0} = \theta_0 + \theta_2 \frac{|V_0|}{|\Omega|} \quad P_{V_1} = \theta_1 + \theta_2 \frac{|V_1|}{|\Omega|} \quad P_{V_2} = \theta_2 \frac{|V_2|}{|\Omega|}.$$

The density function to this mixture distribution can be written as the piecewise constant function

$$f(\mathbf{x}) = \theta_0 f_0(\mathbf{x}) + \theta_1 f_1(\mathbf{x}) + \theta_2 f_2(\mathbf{x}),$$

where

$$f_0(\mathbf{x}) = \frac{1}{|V_0|} \mathbf{1}_{V_0}(\mathbf{x}), \quad f_1(\mathbf{x}) = \frac{1}{|V_1|} \mathbf{1}_{V_1}(\mathbf{x}), \quad f_2(\mathbf{x}) = \frac{1}{|\Omega|}$$

are defined over the entire Ω, with $\mathbf{1}_V(\mathbf{x})$ being an indicator function of region V.

When sampling from V_0 and V_1, to ensure the data are uniformly sampled in those high-dimensional ball regions, we specifically adopt the following algorithm as in [19]

1. Generating random points uniformly on the $(d-1)$-unit sphere.
   (a) Generate a d-dimensional vector $\boldsymbol{x} = (x_1 x_2, \ldots, x_d)$ so that $x_i \sim N(0, 1)$ for $i = 1, 2, \ldots, d$;
   (b) then $\tilde{\boldsymbol{x}} := \frac{\boldsymbol{x}}{||\boldsymbol{x}||_2}$ is a uniformly sampled point form $(d-1)$-unit sphere.
2. Generating a point uniformly at random within the d-ball.
   (a) Let u be a number generated uniformly at random from the interval $[0, 1]$, then $u^{\frac{1}{d}}\tilde{\boldsymbol{x}}$ is a point randomly sampled within the unit ball.
   (b) Further, $r_0 u^{1/d}\tilde{\boldsymbol{x}}$ is a random point in V_0 and $\left((r_T - r_0)u^{1/d} + r_0\right)\tilde{\boldsymbol{x}}$ is a random point in V_1.

To avoid changing the values of the integrals to be evaluated in the loss functions, a piecewise constant factor should be multiplied to the empirical function to correct the approximation of the integrals resulted from a nonuniform distributed training data. For PINN formulation, the empirical loss (4.16) can then be rewritten as

$$\begin{aligned}\mathcal{L}^n_{\text{PINN}} &= \frac{\kappa}{n}\sum_{j=1}^{n} c(\tilde{\mathbf{X}}_j)\left(\partial_t u(T_j, \tilde{\mathbf{X}}_j) - \frac{1}{2}\Delta u(T_j, \tilde{\mathbf{X}}_j)\right)^2 \\ &\quad + \frac{\nu}{n}\sum_{j=1}^{n} c(\tilde{\mathbf{X}}_j)\left(u\left(0, \tilde{\mathbf{X}}_j\right) - u_0\left(\tilde{\mathbf{X}}_j\right)\right)^2 \end{aligned} \tag{5.5}$$

with the correction term

$$c(\mathbf{x}) = \sum_{i=1}^{3} \frac{|V_i|}{|\Omega| P_{V_i}} \mathbf{1}_{V_i}(\mathbf{x}).$$

Here, $\tilde{\mathbf{X}}_j$'s are random points in Ω sampled subject to the aforementioned density function $f(\mathbf{x})$, while T_j will be uniformly sampled from $[0, 1]$. The empirical losses for ϕ formulation and $q-\sigma$ formulation can also be formulated in a similar fashion. The data are then randomly sampled on the fly which are batched into 1000 for each training step. New data is sampled for each batch. Essentially, this sampling scheme enforces data samples in all three regions which can potentially improve the representativity of the training data and thus lead to a faster convergence of the training procedure.

For high-dimensional cases, the initial conditions and PDEs are in fact imposed without the correction factor $c(\mathbf{x})$, i.e., with the efficient sampling scheme, we are essentially minimizing the modified initial/PDE conditions. Taking such terms measured with L^2 as examples, the following loss terms will be minimized

$$\mathcal{L}_I(u) = \int_\Omega (u(0,\mathbf{x}) - u_0(\mathbf{x}))^2 f(\mathbf{x})\, d\mathbf{x},$$
$$\mathcal{L}_{\text{PDE}}(u) = \int_Q \left(\partial_t u - \frac{1}{2}\Delta u^2\right)^2 f(\mathbf{x})\, d\mathbf{x}dt,$$
$$\mathcal{L}_{\partial_t\sigma,\Delta q} = \int_Q (\partial_t\sigma + \Delta q)^2 f(\mathbf{x})\, d\mathbf{x}dt. \tag{5.6}$$

Since $f(\mathbf{x})$ is merely a positive piecewise constant function, such modification will keep the minimizers to these terms unchanged, meaning the desired initial condition and PDE equation will still be imposed with mild assumption on the regularity of the solution ansatz. The reason that the correction constants are not used is because for high-dimensional cases,

$$c(\mathbf{x}) \ll 1\% \quad \forall \mathbf{x} \in V_0 \cup V_1,$$

which means extremely small contribution of samples within these region will be made to update the trainable parameters with SGD. We also notice that while this is possible for imposing the initial condition and PDEs, we must apply $c(\mathbf{x})$ the to inf terms, namely $\mathcal{L}_\phi(\phi)$ and $\mathcal{L}_{q,\sigma}(q,\sigma)$ as the sampling scheme will change the optimization target.

However, the choices of θ_0 and θ_1 still remain to be arbitrary. While numerical examples show that certain choices could lead to faster convergence, there is no clear principle one could follow to make optimal choices. Similar situations happen when we decide the values of ν, κ, γ to balance the terms in the losses. While theoretically these hyper-parameters can be any positive number, the choice of them can heavily influence the training procedure. Some choices seem to help the weighted loss to converge faster compared to others, but there is no justified reason for any certain choices. Therefore, the choices of theses hyper-parameters used in the results reported in this paper are results of trial and error.

The losses are then optimized by tuning the trainable parameters of the neural networks. We take the optimizer that implements the Adam algorithm [15] to train the models. The complete algorithm to establish the loss function and to train the neural networks is implemented using the TensorFlow library [1].

Once the training is finished, to evaluate the quality of the approximate solution obtained with the trained neural networks, we can further quantify the generalization error of it. In particular, we define the relative errors on a solution slice $u(t,x,y,c,\ldots,c)$ at time t for some fixed constant $c \in [-a,a]$, denoting $u(t,x,y,c,\ldots,c)$ by $u(t)$ for simplicity:

$$\begin{aligned} &L^1\textbf{-Relative Error} && \frac{||u_{NN}(t)-u(t)||_1}{||u(t)||_1}, \\ &L^2\textbf{-Relative Error} && \frac{||u_{NN}(t)-u(t)||_2}{||u(t)||_2}, \\ &H^1\textbf{-Relative Error} && \frac{||u_{NN}(t)-u(t)||_{H^1}}{||u(t)||_{H^1}}, \end{aligned} \tag{5.7}$$

where u_{NN} stands for the neural network-based solutions. These norms can be further numerically approximated over a 100×100 evenspaced mesh on $[-a, a]^2$ letting $\{x_i, y_j\}_{i=1,j=1}^{100}$ be the meshgrid points

$$\begin{aligned} ||f(x,y)||_1 &\approx \frac{(2a)^2}{10^4}\sum_{i,j}|f(x_i,y_j)|, \\ ||f(x,y)||_2 &\approx \sqrt{\frac{(2a)^2}{10^4}\sum_{i,j}|f(x_i,y_j)|^2}, \\ ||f(x,y)||_{H^1} &\approx \sqrt{\frac{(2a)^2}{10^4}\sum_{i,j}(|f(x_i,y_j)|^2+|\nabla f(x_i,y_j)|^2)}. \end{aligned}$$

The numerical relative errors can then be computed with predicted values of the neural network solutions evaluated at the mesh-grid points.

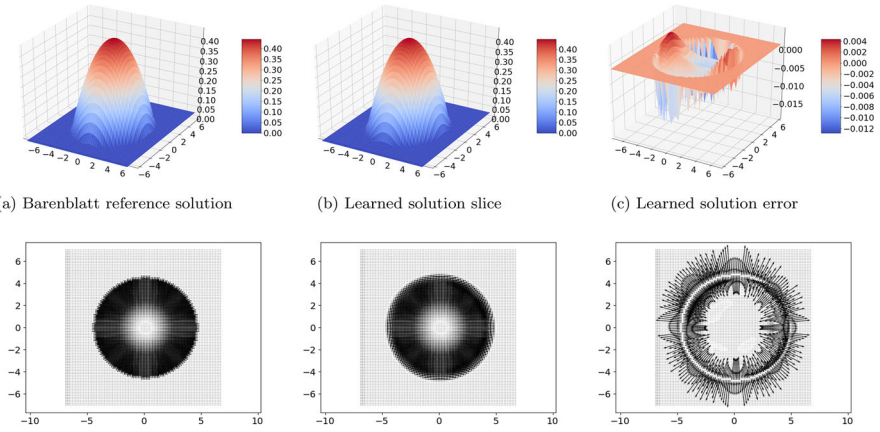
5.2 PINN formulation

In this section, we consider the case, where the QPME (5.3) is solved following a PINN formulation (3.7) in both L^1 and L^2 norm. Specifically speaking, the homogeneous Dirichlet boundary condition is imposed as a hard constraint and the initial condition is imposed as a soft constraint following (4.6). The specific algorithmic settings are further presented in Table 1 along with the relative errors computed for the trained solution slice $u(0.5, x, y, 1.0, \ldots, 1.0; \theta_u^*)$ at time $t = 0.5$ comparing with the exact solution (5.4).

From Table 1, Figs. 2 and 5, one can observe that the PINN formulation can indeed provide numerical solutions that can reasonably approximate the exact ones even in high dimensions. Not only is the neural network able to approximate the function itself, but also the derivative of it. This is essentially a result of successful imposition of the PDE. As can be observed from Figs. 3 and 6, the learned $\partial_t u$ coincides with $\frac{1}{2}\Delta u^2$ which confirms the PDE has been successfully learned. The initial condition can also be softly enforced with term $\mathcal{L}_I$ as training proceeds (See Figs. 4 and 7 for an illustration). The training loss history of PINN is further presented in Fig. 8 term by term. A training convergence can be observed from these plots, which further suggests a convergence to the exact solution ensured by (3.11) and (3.12).

Table 1 PINN formulation (3.3): (hard Dirichlet B.C.+soft I.C.) Relative error comparison for various dimensions

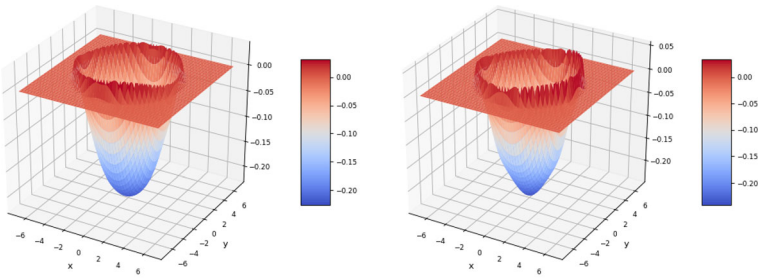
Dimension		1	2	3	4	5	10	15	20	50
Relative error (%) for L^2-PINN	L^2	0.21	0.65	0.61	0.55	1.1	0.86	1.72	5.50	16.03
	L^1	0.14	0.4	0.39	0.43	0.94	0.71	1.64	5.12	15.26
	H^1	4.28	8.25	7.93	7.45	8.64	8.09	9.88	10.87	28.5
Relative error (%) for L^1-PINN	L^2	0.45	0.78	0.95	1.27	2.07	4.86	10.46	9.73	10.76
	L^1	0.23	0.5	0.69	1.1	1.91	4.65	8.91	8.37	10.47
	H^1	5.11	8.98	9.38	9.51	10.48	16.57	16.45	13.13	21.55
Formulation weight	ν	10^3					1			
	κ	1					10^3			
NN architecture	# trainable	41,001	41,201	41,401	41,601	41,801	42,801	43,801	169,601	181,601
	Width/Depth	200/2							400/2	
Data sampling	θ_0	0.3								
	θ_1	0.3							0.2	
Training	Steps	10^5							2×10^5	
	Learning rate	10^{-3}								



(a) Barenblatt reference solution (b) Learned solution slice (c) Learned solution error

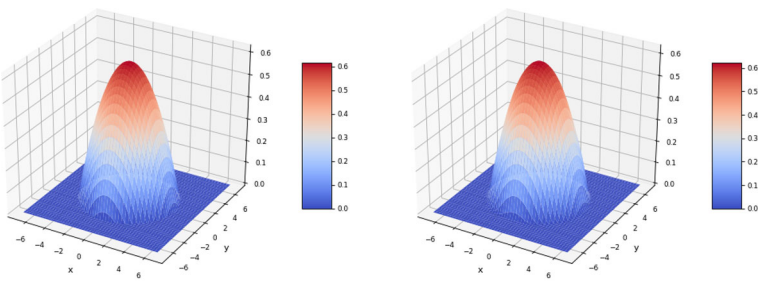
(d) Barenblatt reference solution gradient (e) Learned solution gradient (f) Learned solution gradient error

Fig. 2 15D, L^2- PINN formulation (3.7) Predicted solution slice $u(0.5, x, y, 1.0, \ldots, 1.0)$ for $\mathbf{x} \in \Omega = [-7, 7]^{15}$, $t = 0.5$



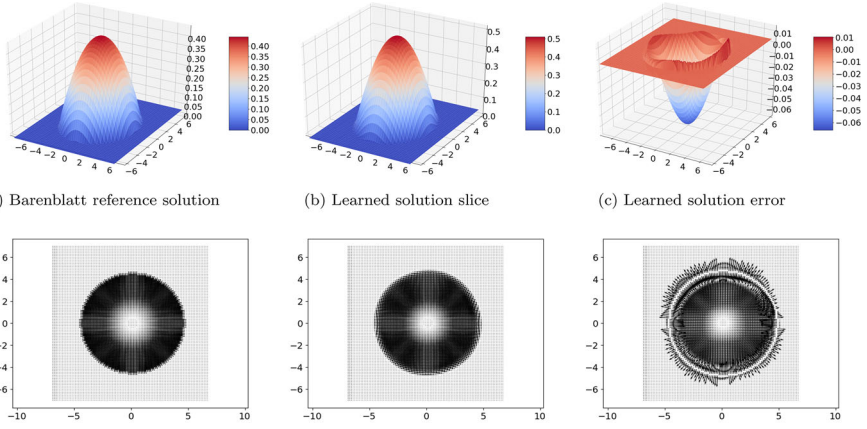
(a) Learned u_t (b) Learned $\frac{1}{2}\Delta u^2$

Fig. 3 15D, L^2- PINN formulation (3.7), predicted partial derivatives



(a) Exact u_0 (b) Learned initial value

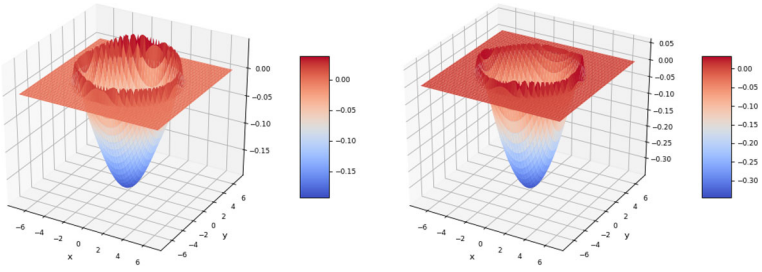
Fig. 4 15D, L^2- PINN formulation (3.7), predicted initial value $u(0, x, y, 1.0, \ldots, 1.0)$ for $\mathbf{x} \in \Omega = [-7, 7]^{15}$



(a) Barenblatt reference solution (b) Learned solution slice (c) Learned solution error

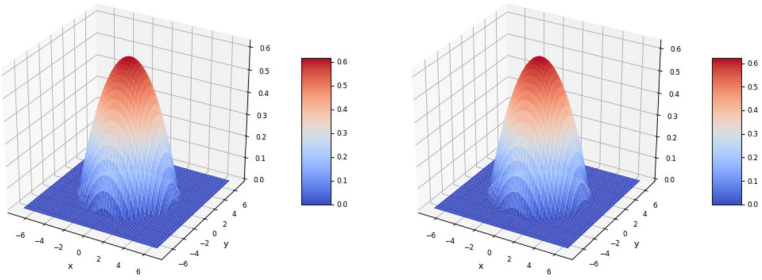
(d) Barenblatt reference solution gradient (e) Learned solution gradient (f) Learned solution gradient error

Fig. 5 15D, L^1- PINN formulation (3.7): Predicted solution slice $u(0.5, x, y, 1.0, \ldots, 1.0)$ for $\mathbf{x} \in \Omega = [-7, 7]^{15}$, $t = 0.5$



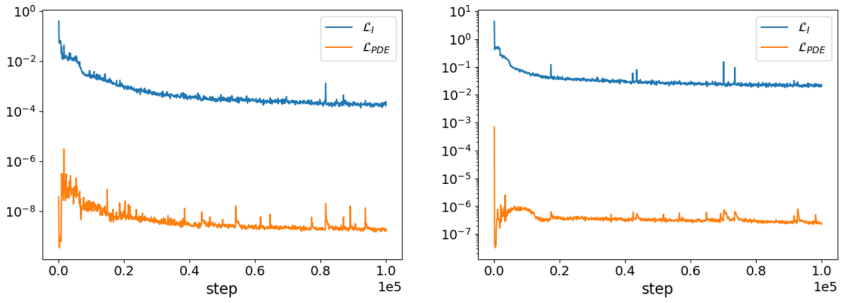
(a) Learned u_t (b) Learned $\frac{1}{2}\Delta u^2$

Fig. 6 15D, L^1- PINN formulation (3.7): predicted partial derivatives



(a) Exact u_0 (b) Learned initial value

Fig. 7 15D, L^1- PINN formulation (3.7), predicted initial value $u(0, x, y, 1.0, \ldots, 1.0)$ for $\mathbf{x} \in \Omega = [-7, 7]^{15}$



(a) L^2-PINN (b) L^1-PINN

Fig. 8 15D: Training loss history by term

In addition, one can observe that learning the solution to QPME does not require the number of the trainable parameters of the solution ansatz to scale exponentially, which, in contrast to mesh-based solvers, is advantageous in dealing with high-d problems.

However, from Table 1, one can observe that the generalization errors of the neural network-based solutions of high-dimensional cases are larger compared to that of low-dimensional cases. This could be a result of using larger neural network to approximate a more complicated solution in high-dimensional cases. Numerical experiments show that neural networks with width 200 is no longer sufficient in approximating solutions to QPME with dimension larger than 15, thus a larger network was adopted. Such network naturally requires more training and data for convergence. Since the training steps (data) was not quadrupled as the number of trainable does, which could have contributed to a larger approximation error. In addition, whether assigning quadrupled training steps and training data could bring significant improvement on the approximation accuracy is also questionable as the optimization of θ_u is highly nonconvex, which means one has to accept a significant and unavoidable uncertainty about optimization success with SGD or its variants.

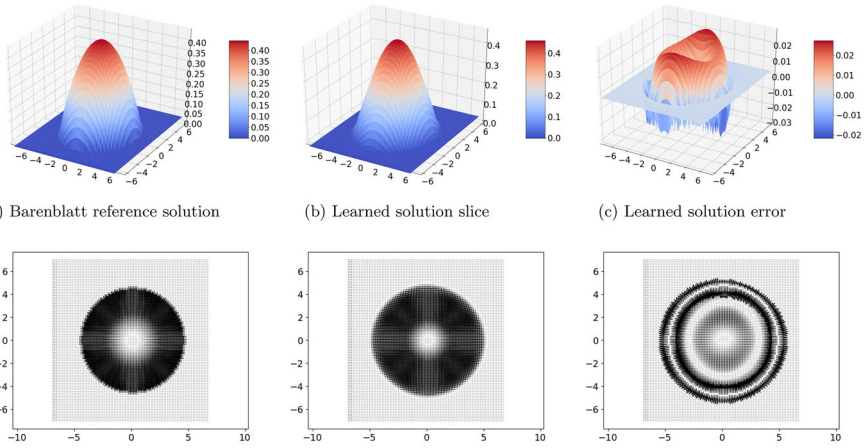
5.2.1 ϕ formulation

In this section, we consider the ϕ formulation (3.25) to solve QPME (5.3) in both L^1 and L^2 norm. The homogeneous Dirichlet boundary condition is enforced as a hard constraint following (4.9). The initial condition can also be enforced softly with term $\mathcal{L}_I$ similar to the PINN formulation. The specific algorithmic settings are further presented in Table 2 along with the relative errors computed for the trained solution slice $u(0.5, x, y, 1.0, \ldots, 1.0; \theta_u^*)$ at time $t = 0.5$ comparing with the exact solution (5.4).

From Table 2, Figs. 9 and 10, one can observe that the ϕ formulation can indeed provide numerical solutions that closely approximate the exact ones up till dimension 20. Not only is the neural network able to provide a reasonable approximation of the function itself, but also the derivatives of it. The mismatch mainly concentrated near the region where the solution is not smooth (free boundary). The predicted minimizer ϕ to (3.25) is also presented as in Fig. 11.

Table 2 ϕ formulation (3.19) (hard Dirichlet B.C.+soft I.C.): Relative error comparison for various dimensions

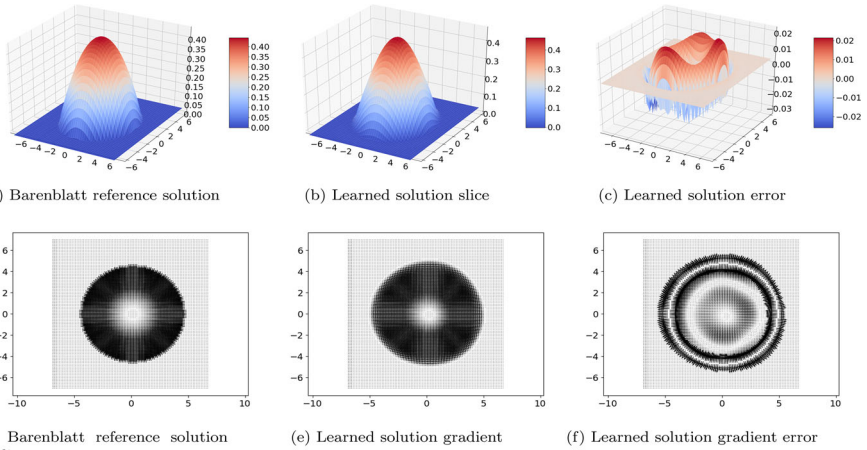
Dimension		1	2	3	4	5	10	15	20	50
Relative errors (%) for $L^2-\phi$ formulation	L^2	3.58	4.95	4.41	9.77	5.77	3.82	8.29	14.45	54.26
	L^1	3.23	5.87	4.57	9.98	6.22	3.97	8.99	15.56	77.65
	H^1	16.44	18.84	20.18	27.29	16.97	14.98	19.21	24.15	71
Relative errors (%) for $L^1-\phi$ formulation	L^2	2.32	5.01	5.06	8.8	4.87	3.62	9.25	26.67	52.24
	L^1	2.11	6.16	5.84	9.21	4.84	3.52	9.49	28.05	85.13
	H^1	14.52	25.25	18.23	25.73	16.82	13.73	20.3	44.74	67.49
Formulation weights	ν	10^3				1				
	κ	1				10^3		10^4	10^5	10^3
NN architecture	Width/Depth	200/2							400/2	
	# trainable	41,001	41,201	41,401	41,601	41,801	42,801	43,801	169,601	181,601
Data sampling	θ_0	0.3							0.2	0.4
	θ_1	0.3								0.4
Training	Steps	10^5							6×10^5	2×10^5
	Learning rate	10^{-3}								10^{-4}



(a) Barenblatt reference solution (b) Learned solution slice (c) Learned solution error

(d) Barenblatt reference solution gradient (e) Learned solution gradient (f) Learned solution gradient error

Fig. 9 15D, $L^2 - \phi$ formulation (3.19): Predicted solution slice $u(0.5, x, y, 1.0, \ldots, 1.0)$ for $\mathbf{x} \in \Omega = [-7, 7]^{15}$, $t = 0.5$

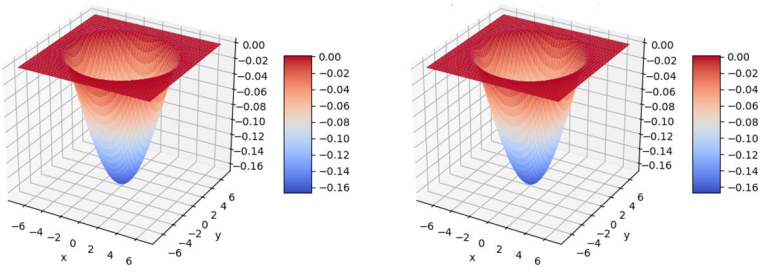


(a) Barenblatt reference solution (b) Learned solution slice (c) Learned solution error

(d) Barenblatt reference solution gradient (e) Learned solution gradient (f) Learned solution gradient error

Fig. 10 15D, $L^1 - \phi$ formulation (3.19): Predicted solution slice $u_\phi(0.5, x, y, 1.0, \ldots, 1.0)$ with for $\mathbf{x} \in \Omega = [-7, 7]^{15}$, $t = 0.5$

Theoretically speaking, compared to PINN, ϕ formulation is advantageous as it can be applied to solve a wider range of QPMEs, whose solution are less regular or smooth. However, for the case being tested, we do encounter more challenges in the training process especially for the high-dimensional cases compared with that of PINN. One observation is that the generalization error of the testing solution slice gets larger as the dimension gets higher. This could attributes to the nature of the exact solution U_2 as its nonzero region only accounts for a tiny small portion of Ω ($\ll 1‰$) when d is large. That is to say, the zero function will be a pretty good approximation of U_2 already in both $L^1(\Omega)$ and $L^2(\Omega)$ sense. The training can thus be easily trapped in a



(a) Learned through L^1-ϕ formulation (b) Learned through L^2-ϕ formulation

Fig. 11 15D: Predicted $\phi(0.5, x, y, 1.0, \ldots, 1.0; \theta_\phi^*)$

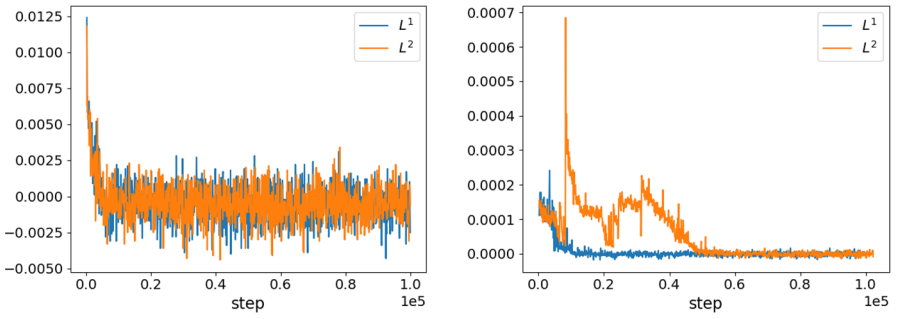
local minimum $u_\phi = 0$ which can be reached by a neural network $\phi = 0$. In addition, the generalization error reported measures the error for a solution slice projected onto a two-dimensional space in stead of in Ω to ease the computation and visualization, which can be an uncomprehensive measurement of the error. Moreover, the selected slice is a slice whose values are dominated by nonzero ones, which can also be an unfair representative of the entire solution to quantify the relative error.

The reason that PINN formulation seems to suffer less from this effects is probably the application of efficient sampling. Since the correction term $c(\mathbf{x})$ is not applied for any terms in the loss functional of PINN, meaning a very large weight was put on the region where the solution is nonzero when evaluating $\mathcal{L}_{\text{PINN}}$, which could have helped the solution ansatz to escape from the local minimum. However, by the nature of ϕ formulation, the $c(\mathbf{x})$ can not be omitted. Otherwise, the target functional will be changed entirely.

Furthermore, while we are able to identify the desired solutions in many cases, theoretically, one can not guarantee meaningful solutions to QPME form the training of ϕ formulation. In fact, both the condition $1 - \Delta\phi$ and $u_\phi \geq 0$ are not enforced at all in this formulation. Theses conditions can only be used post training to carry out a solution selection or as a criteria for early truncation of training.

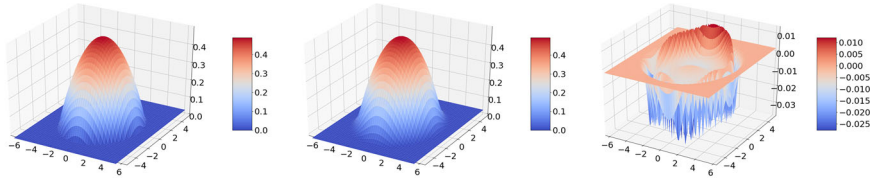
Artificial choices of other algorithmic ingredients such as batch sizes, learning rates, θ_0 and θ_1 will also inevitable influence the optimization process providing limited computational resources.

We further observe that the optimization of $\mathcal{L}_\phi(u_\phi(t, \mathbf{x}; \theta_\phi))$ indeed converges to $-\int_Q U_2^2 \, d\mathbf{x}dt$ as training proceeds with u_ϕ being the parametrize solution ansatz as stated in (4.8). This observation in fact confirms the theoretical result (3.21) derived in [5]. In Fig. 12, we specifically use the batch of training data at each training step to evaluate empirically the value of $-\int_Q U_2^2 \, d\mathbf{x}dt$ for the exact solution $U_2(t, \mathbf{x})$ defined as in (5.4), and further compare it with the empirical loss $\mathcal{L}_\phi(u_\phi)$ based on the neural network solution u_ϕ at that time. As one can observe, the difference of the two values gradually reduces as the training continues, which verifies the training effectiveness of this formulation.



(a) **5D** (b) **10D**

Fig. 12 Empirical $\int_Q U_2^2(t,\mathbf{x})d\mathbf{x}dt + \mathcal{L}_\phi\big(u_\phi(t,\mathbf{x};\theta_\phi)\big)$ as training proceeds



(a) Barenblatt reference solution (b) Learned solution slice (c) Learned solution error

Fig. 13 10D, $L^2 - q - \sigma$ formulation (3.30): Predicted solution slice $u(0.5, x, y, 1.0, \ldots, 1.0)$ for $\mathbf{x} \in \Omega = [-6, 6]^{10}$, $t = 0.5$

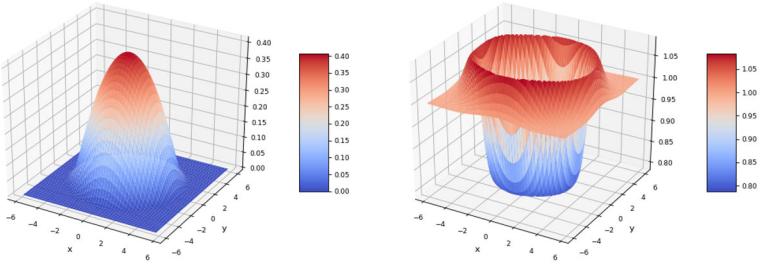
5.3 q-σ formulation

Since $q - \sigma$ formulation (3.30) is developed based on the ϕ formulation, the training thus also suffers from challenges met in the training of ϕ formulation, i.e., the training can be easily trapped in a local minimum, i.e., $u_{q,\sigma} = 0$ for high-dimensional cases. Additionally, the partial derivatives of ϕ are separated into two independent functions q and σ, whose correlation is only enforced softly with the loss term $\mathcal{L}_{\partial_t\sigma,\Delta q}$, which can pose more challenges to the optimization of the target functional. Additional hyper-parameter γ is also introduced to adjust the weight of $\mathcal{L}_{\partial_t\sigma,\Delta q}$ whose optimal choice is again obscure. Due to such reasons, only results for dimension 1 to 10 are reported as no reasonable results for higher-dimensions were obtained in the scope of experiments that were carried out.

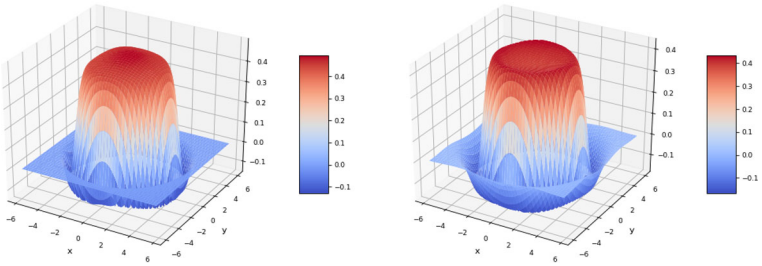
Specifically, the homogeneous Dirichlet boundary condition is imposed as a hard constraint following (4.15) and the condition for σ is imposed with (4.11). The condition (4.12) was not strongly imposed for training reasons. The initial condition can then be softly enforced with term $\mathcal{L}_I$ as mentioned earlier. The specific algorithmic settings are further presented in Table 3 along with the relative errors computed for the trained solution slice $u_{q,\sigma}(0.5, x, y, 1.0, \ldots, 1.0; \theta_q, \theta_\sigma)$ at time $t = 0.5$ comparing with the exact solution (5.4). The comparison of predicted solutions with exact solution are further presented in Figs. 13 and 15. In addition, the predicted function q and σ are depicted in Figs. 14 and 16. These figures are further used to show the

Table 3 $q-\sigma$ formulation (3.24) (hard Dirichlet B.C.+soft I.C.): Relative error comparison for various dimensions

Dimension		1	2	3	4	5	10
Relative errors (%) for $L^2-q-\sigma$ formulation	L^2	1.95	3.2	3.88	3.97	4.77	4.03
	L^1	1.64	3.11	3.5	4.02	5.11	4.14
Relative errors (%) for $L^1-q-\sigma$ formulation	L^2	2.06	2.96	3.83	3.94	5.63	4.28
	L^1	1.72	2.79	3.28	3.72	6.41	4.59
Formulation weights	ν	1					
	κ	10^3					
	γ	10^3					1
NN architecture	Width/Depth	200/2					
	# trainable	82,002	82,402	82,802	83,202	83,602	85,602
Data sampling	θ_0	0.3					
	θ_1	0.3					
Training	Steps	10^5					
	Learning rate	10^{-3}					

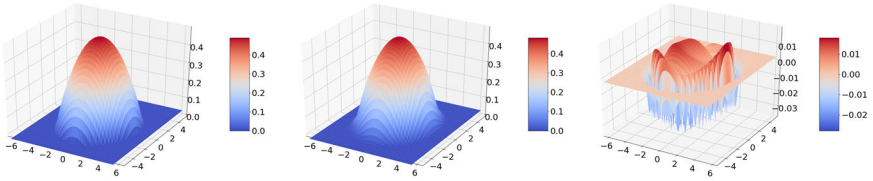


(a) Learned q (b) Learned σ



(c) Learned $-\Delta q$ (d) Learned $\partial_t \sigma$

Fig. 14 10D, $L^2 - q - \sigma$ formulation (3.30): predicted q, σ and their partial derivatives



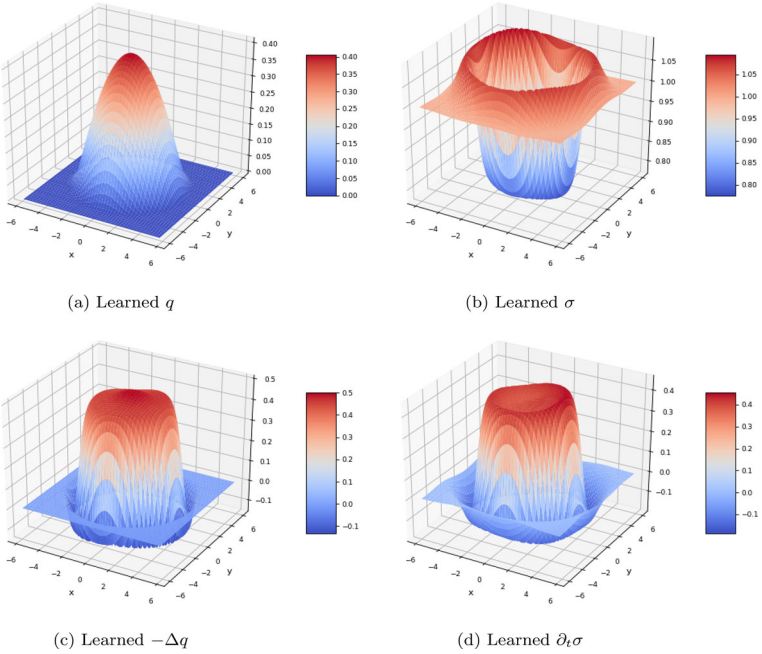
(a) Barenblatt reference solution (b) Learned solution slice (c) Learned solution error

Fig. 15 10D, $L^1 - q - \sigma$ formulation (3.30): Predicted solution slice $u(0.5, x, y, 1.0, \ldots, 1.0)$ for $\mathbf{x} \in \Omega = [-6, 6]^{10}$, $t = 0.5$

predicted $-\Delta q$ and $\partial_t \sigma$ to verify that the condition

$$\Delta q + \partial_t \sigma = 0$$

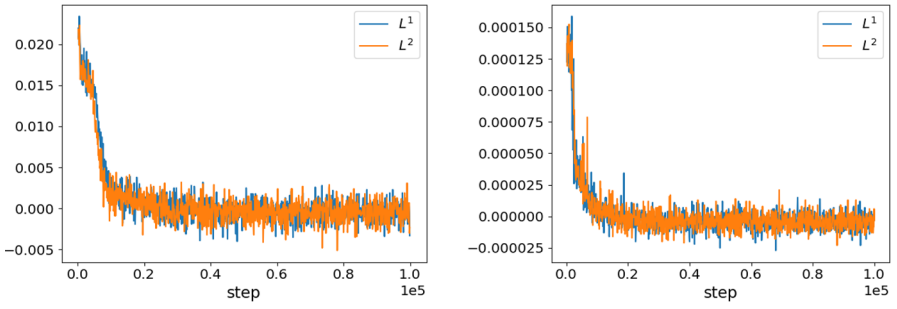
is satisfied. Finally, Fig. 17 is used to demonstrate the computational value of $\mathcal{L}_{q,\sigma}$ converges to $-\int_Q U_2^2 \, d\mathbf{x}dt$ as training proceeds. This observation confirms once again the theoretical result (3.21) derived in [5]. Here, the batch of training data at each training step is used to evaluate empirically the value of $-\int_Q U_2^2 \, d\mathbf{x}dt$ for the exact solution $U_2(t, \mathbf{x})$ defined as in (5.4). Such value is further compared with the empirical loss $\mathcal{L}_{q,\sigma}(u_{q,\sigma})$ at that time. The difference of the these values gradually reduces as the training continues (Fig. 17).



(a) Learned q (b) Learned σ

(c) Learned $-\Delta q$ (d) Learned $\partial_t \sigma$

Fig. 16 10D, $L^1 - q - \sigma$ formulation (3.30): predicted q, σ and their partial derivatives



(a) **5D** (b) **10D**

Fig. 17 Empirical $\int_Q U_2^2(t, \mathbf{x}) d\mathbf{x} dt + \mathcal{L}_{q,\sigma}\left(u_{q,\sigma}(t, \mathbf{x}; \theta_q, \theta_\sigma)\right)$ v.s. training steps

6 Numerical example: waiting-time phenomena

In this section, we further consider the following IBVP

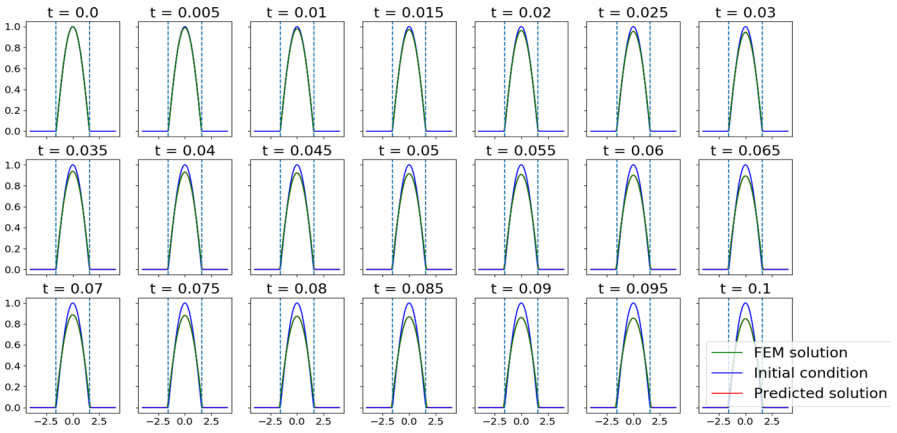
$$\begin{aligned} &\partial_t u = \frac{1}{2}\Delta u^2 \quad (t, \mathbf{x}) \in Q = [0, 1] \times \Omega, \\ &u(0, \mathbf{x}) = u_0(\mathbf{x}) = \begin{cases} \cos(|\mathbf{x}|), & |\mathbf{x}| \leq \frac{\pi}{2}, \\ 0, & \text{elsewhere}, \end{cases} \\ &u(t, \mathbf{x})|_{\partial\Omega} = 0, \end{aligned} \tag{6.1}$$

where $\Omega = [-4, 4]^d$. The general exact solution to (6.1) can hardly be derived. When $d = 2$, the reference solution can be taken to be the numerical solutions obtained with a moving mesh finite element method following [29] instead. In particular, the mesh is advanced forward in time using a forward Euler time-stepping scheme. These mesh-based results are then compared with the ones obtained following a deep learning framework under various formulations. For higher dimensions, the mesh-based solver in general will suffer from curse of dimensionality. The moving mesh method would also be more challenging to design. Therefore, for comparison reasons, we only present the results for $d = 2$ while noticing the higher dimensional cases can also be handled by the neural network-based algorithms.

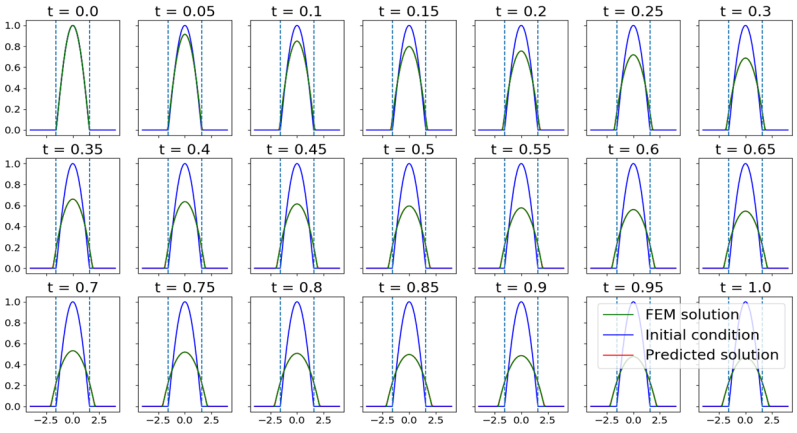
We also note that the solution to PME of this type of initial condition exhibits a waiting-time phenomenon [21]. In fact, the velocity of the free boundary of QPME is given by Darcy's law [24], i.e.,

$$\Gamma'(t) = \lim_{\mathbf{x} \to \Gamma(t)^-} \nabla \left(\frac{u^2}{2} \right), \tag{6.2}$$

where the limit is taken from the interior of the support. Thus, as one may compute, the free boundary of solution to (6.1) should not move until a finite amount of time has elapsed as initially $\Gamma'(0) = \nabla \left(\frac{u_0^2}{2} \right)$ vanishes at the free boundary $\Gamma(0) : |\mathbf{x}| = \frac{\pi}{2}$. This phenomena of waiting can be observed from solutions obtained with Finite Element Method as shown in Fig. 18, where the dashed vertical lines indicate the exact initial location of the free boundary. In Fig. 18a, a series of snapshots of the solution for $t \in [0, 0.1]$ is plotted, while in 18b, the solution snapshots for a broader range of time is plotted. As one may observe, the free boundary of the solution barely moves in the entire time of $t \in [0, 0.1]$ and only start to change by the time of $t = 0.2$. This phenomena can also be accurately captured by a solution obtained following the PINN formulation (3.7). In specific, the solution obtained with the PINN formulation is presented in comparison with the ones obtained with the moving mesh FEM in Fig. 18. The solution slices essentially overlap one another. Parallelly, problem (6.1) is also solved with the ϕ formulation (3.22) and the $q - \sigma$ formulation (3.24). The comparison of the resulted solution with a FEM solution is then presented as in Figs. 19 and 20, respectively, which verifies the effectiveness of the neural network-based solutions using ϕ formulation and $q - \sigma$ formulation.



(a) Snapshot solution slices for $t \in [0, 0.1]$

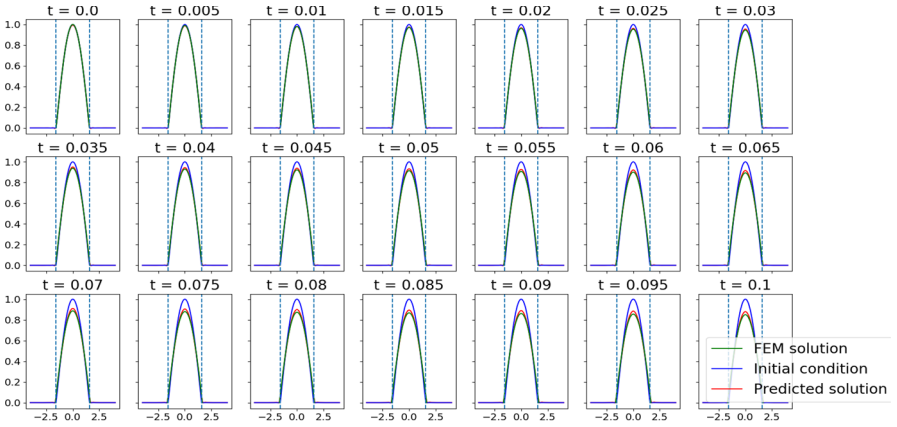


(b) Snapshot solution slices for $t \in [0, 1.0]$

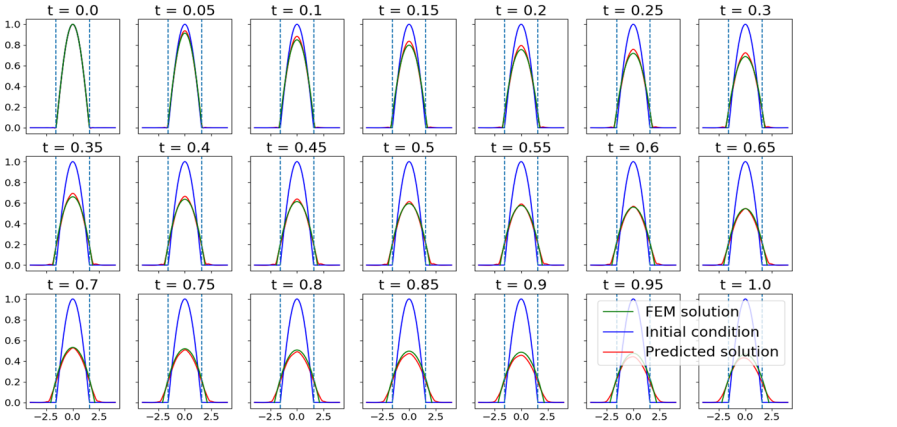
Fig. 18 2d, L^2- PINN formulation, waiting-time phenomena (6.1): snapshots of solution cross-section $u(t, x, 0)$ at $y = 0$. Green: reference solutions obtained with moving mesh FEM ($DOF = 901$); red: predicted solutions obtained with PINN formulation; blue: initial condition. (Colour figure online)

7 Conclusion

In this paper, we explored different variational forms in solving high-dimensional QPME with neural networks. In specific, three formulations were considered. For the PINN formulation, the solution is directly parametrized and the PDE residual is minimized. A theoretical analysis is carried out to show that the convergence of the PINN loss guarantees a convergence to the exact solution in L^1 sense. Moreover, this analysis also suggests the use of the L^1 norm to quantify the residual and approximation



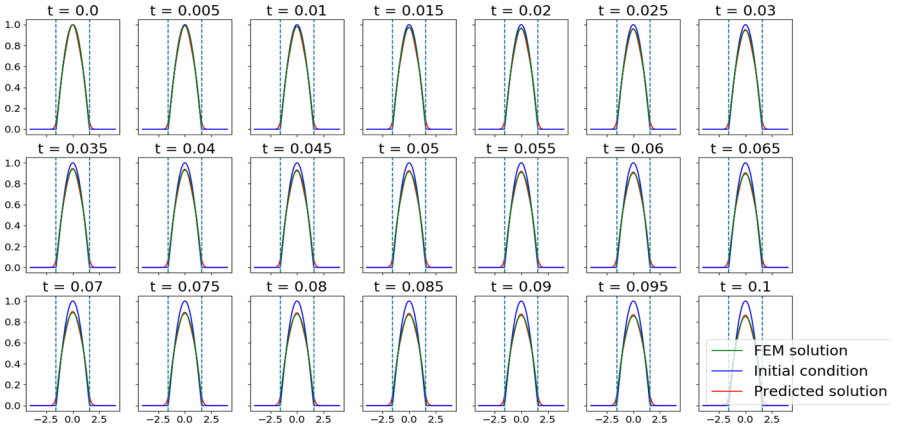
(a) $t \in [0, 0.1]$



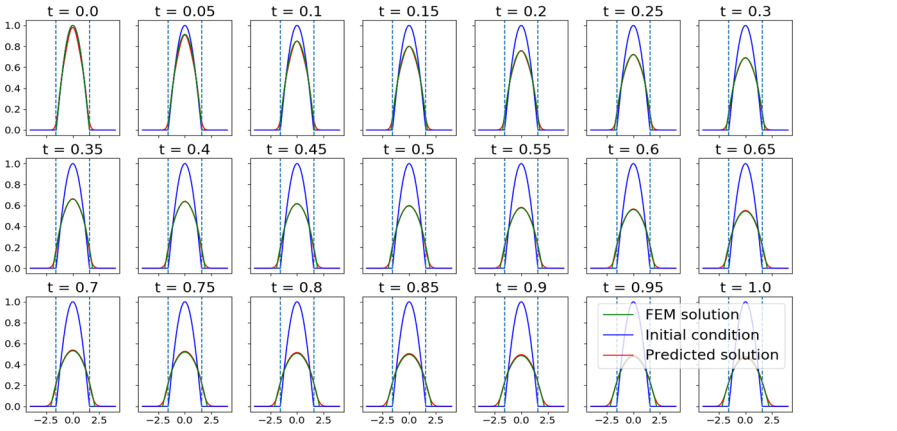
(b) Snapshot solution slices for $t \in [0, 1.0]$

Fig. 19 2d, $L^2-\phi$ formulation, waiting-time phenomena (6.1): snapshots of solution cross-section $u(t, x, 0)$ at $y = 0$. Green: reference solutions obtained with moving mesh FEM ($DOF = 901$); red: predicted solutions obtained with ϕ formulation; blue: initial condition. (Colour figure online)

error. In addition, inspired by the work [5], a ϕ formulation and a $q - \sigma$ formulation are further presented and used to solve the QPME in a very weak sense. Theoretically, these formulation can identify solutions with less regularity. All formulations are then tested with the Barrenbaltt solution in low and high dimensions. Experiments have shown that ϕ formulation and q, σ formulation can provide approximate solutions with a similar level of accuracy compared with PINN in low-dimensional cases but the optimization aspect continues to pose challenges in high-dimensional cases. A two-dimensional example of QPME that exhibits waiting phenomena is also presented to



(a) $t \in [0, 0.1]$



(b) Snapshot solution slices for $t \in [0, 1.0]$

Fig. 20 2d, $L^2 - q - \sigma$ formulation, waiting-time phenomena (6.1): snapshots of solution cross-section $u(t, x, 0)$ at $y = 0$. Green: reference solutions obtained with moving mesh FEM ($DOF = 901$); red: predicted solutions obtained with $q - \sigma$ formulation; blue: initial condition. (Colour figure online)

show the capability of deep learning-based methods in identifying solution features as such.

Other aspects of the discussion toward solving QPME with deep learning includes the hard and soft imposition of certain conditions of the solutions in all formulations such as initial conditions and boundary conditions. Additionally, an efficient sampling scheme is proposed aiming at a faster convergence toward the solution desired especially in high-dimensional cases. These treatments in principal can also be applied in other scenarios where the PDE solutions are parametrize with neural networks.

While such efforts can all contribute to more efficient implementation of solving high-dimensional QPMEs, we must admit that the training success is overwhelmed by the large number of hyper-parameters. Moreover, for practical applications, neural network training using stochastic gradient descent type schemes, which means one must accept a significant and unavoidable uncertainty about optimization success. An efficient strategy on making choices of hyper-parameters could potentially be an interesting direction for future investigations. More broadly, whether a similar variational form could be derived for general m of porous medium equation is also an open question.

Funding This work is supported in part by National Science Foundation via grant DMS- 2012286 and by Department of Energy via grant DE-SC0019449.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: {TensorFlow}: a system for {Large-Scale} machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283 (2016)
2. Barron, A.R.: Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans. Inf. Theory **39**(3), 930–945 (1993)
3. Bing, Yu., et al.: The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. Commun. Math. Stat. **6**(1), 1–12 (2018)
4. Boelens, A.M., Venturi, D., Tartakovsky, D.M.: Parallel tensor methods for high-dimensional linear PDEs. J. Comput. Phys. **375**, 519–539 (2018)
5. Brenier, Y.: Examples of hidden convexity in nonlinear PDEs (2020)
6. Dektor, A., Venturi, D.: Dynamically orthogonal tensor methods for high-dimensional nonlinear PDEs. J. Comput. Phys. **404**, 109125 (2020)
7. Dektor, A., Rodgers, A., Venturi, D.: Rank-adaptive tensor methods for high-dimensional nonlinear PDEs. J. Sci. Comput. **88**(2), 1–27 (2021)
8. Dolgov, S., Kalise, D., Kunisch, K.K.: Tensor decomposition methods for high-dimensional Hamilton-Jacobi-Bellman equations. SIAM J. Sci. Comput. **43**(3), A1625–A1650 (2021)
9. Eigel, M., Neumann, J., Schneider, R., Wolf, S.: Non-intrusive tensor reconstruction for high-dimensional random PDEs. Comput. Methods Appl. Math. **19**(1), 39–53 (2019)
10. Han, J., Jentzen, A., et al.: Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Commun. Math. Stat. **5**(4), 349–380 (2017)
11. Han, J., Jentzen, A., Weinan, E.: Solving high-dimensional partial differential equations using deep learning. Proc. Natl. Acad. Sci. **115**(34), 8505–8510 (2018)
12. Henry-Labordere, P.: Counterparty risk valuation: a marked branching diffusion approach. Available at SSRN 1995503 (2012)
13. Henry-Labordere, P., Oudjane, N., Tan, X., Touzi, N., Warin, X.: Branching diffusion representation of semilinear PDEs and Monte Carlo approximation. In: Annales de l'Institut Henri Poincaré, Probabilités et Statistiques, vol. 55, pp. 184–210. Institut Henri Poincaré (2019)
14. Jentzen, A., Salimova, D., Welti, T.: A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. arXiv preprint arXiv:1809.07321 (2018)
15. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
16. Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Netw. **9**(5), 987–1000 (1998)
17. Leibenzon, L.S.: The motion of a gas in a porous medium. Complete Works, Vol. 2, Acad. Sci. URSS Moscow (Russian) **63**, 8–9 (1930)

18. Lu, J., Lu, Y., Wang, M.: A priori generalization analysis of the deep ritz method for solving high dimensional elliptic partial differential equations. In: Conference on Learning Theory, pp. 3196–3241. PMLR (2021)
19. Marsaglia, G.: Choosing a point from the surface of a sphere. Ann. Math. Stat. **43**(2), 645–646 (1972)
20. Muskat, M.: The flow of homogeneous fluids through porous media. Soil Sci. **46**(2), 169 (1938)
21. Ngo, C., Huang, W.: A study on moving mesh finite element solution of the porous medium equation. J. Comput. Phys. **331**, 357–380 (2017)
22. 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**, 686–707 (2019)
23. Richter, L., Sallandt, L., Nüsken, N.: Solving high-dimensional parabolic PDEs using the tensor train format. In: International Conference on Machine Learning, pp. 8998–9009. PMLR (2021)
24. Shmarev, S.: Interfaces in solutions of diffusion-absorption equations in arbitrary space dimension, pp. 257–273. Trends in Partial Differential Equations of Mathematical Physics (2005)
25. Shukla, M.B., Kok, R., Prasher, S.O., Clark, G., Lacroix, R.: Use of artificial neural networks in transient drainage design. Trans. ASAE **39**(1), 119–124 (1996)
26. Vázquez, J.L.: The Porous Medium Equation: Mathematical Theory. Oxford University Press, Oxford (2007)
27. Warin, X.: Variations on branching methods for non linear PDEs. arXiv preprint arXiv:1701.07660 (2017)
28. Warin, X.: Nesting Monte Carlo for high-dimensional non-linear PDEs. Monte Carlo Methods Appl. **24**(4), 225–247 (2018)
29. Wells, B.V.: A moving mesh finite element method for the numerical solution of partial differential equations and systems. PhD thesis, Citeseer (2004)
30. Zang, Y., Bao, G., Ye, X., Zhou, H.: Weak adversarial networks for high-dimensional partial differential equations. J. Comput. Phys. **411**, 109409 (2020)