SMAI-JCM
SMAI Journal of Computational Mathematics

Improving Weak PINNs for Hyperbolic Conservation Laws: Dual Norm Computation, Boundary Conditions and Systems

Aidan Chaumet & Jan Giesselmann

Volume 10 (2024), p. 373-401.
https://doi.org/10.5802/smai-jcm.116

e-ISSN: 2426-8399

SMAI Journal of Computational Mathematics
Vol. 10, 373-401 (2024)

Improving Weak PINNs for Hyperbolic Conservation Laws: Dual Norm Computation, Boundary Conditions and Systems

Aidan Chaumet [1]
Jan Giesselmann [2]

[1] Department of Mathematics, Technical University of Darmstadt, Germany
E-mail address: chaumet@mathematik.tu-darmstadt.de
[2] Department of Mathematics, Technical University of Darmstadt, Germany
E-mail address: giesselmann@mathematik.tu-darmstadt.de.

Abstract. We consider the approximation of entropy solutions of nonlinear hyperbolic conservation laws using neural networks. We provide explicit computations that highlight why classical PINNs will not work for discontinuous solutions to nonlinear hyperbolic conservation laws and show that weak (dual) norms of the PDE residual should be used in the loss functional. This approach has been termed "weak PINNs" recently. We suggest some modifications to weak PINNs that make their training easier, which leads to smaller errors with less training, as shown by numerical experiments. Additionally, we extend wPINNs to scalar conservation laws with weak boundary data and to systems of hyperbolic conservation laws. We perform numerical experiments in order to assess the accuracy and efficiency of the extended method.

2020 Mathematics Subject Classification. 35L65, 65M99.

Keywords. physics-informed learning, PINNs, hyperbolic conservation laws, entropy solution.

1. Introduction

Machine learning is an effective approach for a variety of challenging applications such as image generation, computer vision or natural language processing. Their ubiquitous success has triggered mathematical studies into the use of neural networks for approximating solutions to partial differential equations. Physics informed neural networks (PINNs) as introduced in [37] are a popular approach for approximating PDEs. Classical PINNs leverage automatic differentiation for neural networks to compute the PDE residual and minimize its L^2 norm in order to approximate solutions. For problems with smooth solutions, error estimates such as [6, 33] explain the success of PINNs. In contrary, empirical evidence suggests that classical PINNs do not work for discontinuous solutions [31] that naturally occur as solutions to nonlinear hyperbolic conservation laws.

Hyperbolic conservation laws arise in many models in continuum physics. Examples include the Euler- and shallow water equations. The development of numerical schemes for hyperbolic conservation laws in up to three space dimensions is quite mature and PINNs are non-competitive for many classical applications. However, there are new challenges that require schemes which are efficient in high space dimensions, where mesh-based methods are unfeasible, e.g. the approximation of correlation measures that appear in the definition of statistical solutions [13].

The first goal of this manuscript is to explain the failure of PINNs for discontinuous solutions of nonlinear hyperbolic conservation laws using explicit computations. This establishes that the empirical

The authors gratefully acknowledge the computing time provided to them on the high-performance computer Lichtenberg at the NHR Centers NHR4CES at TU Darmstadt. This is funded by the Federal Ministry of Education and Research, and the state governments participating on the basis of the resolutions of the GWK for national high performance computing at universities.
https://doi.org/10.5802/smai-jcm.116

observations are not due to practical difficulties, such as insufficiently large networks or lack of training. Instead, we show that classical PINNs are fundamentally unable to approximate discontinuous solutions of nonlinear hyperbolic conservation laws due to their loss functional. Several elementary, but to the best of our knowledge new, computations show that good continuous approximations to discontinuous solutions do not produce residuals which are small in the L^p norm for any $1 \leq p \leq \infty$. Thus, minimizing the L^p norm of the PDE residual over the set of functions represented by a neural network cannot give good approximations.

Our computations show that instead, for good approximations, weak norms of the PDE residual are small. This is analogous to replacing the strong formulation of PDEs with their weak formulation to describe discontinuous solutions. Examples of PINNs using weak norms of PDE residuals to successfully approximate discontinuous solutions are variational PINNs [22] or weak PINNs [7] (wPINNs). Both of these approaches approximate weak norms of the PDE residual by invoking the sup-based definition of weak norms and maximizing over test functions. This gives a saddle-point problem during training. The loss is minimized with respect to the approximate solution and maximized with respect to the test function giving the weak norm of the PDE residual. Training this type of PINNs is difficult, especially because the maximization problem may get stuck in bad local maxima. This also limits their attractiveness for concrete applications.

This motivates the second goal of this paper. In order to make training wPINNs easier, we approach the weak norm estimation differently. Instead of using the sup-based definition of weak norms, we identify the maximizer of the dual pairing as the solution to a family of dual elliptic problems. Then, the solutions to the elliptic problems are identified as maximizers of a concave functional. We use a neural network to represent the solution and train it with gradient ascent. This is similar to the Deep Ritz Method for solving the Poisson problem [11] and accelerates the learning of the dual problem, leading to overall more stable and effective learning of wPINNs.

For many practical applications, further generalizations of wPINNs are required. We discuss extensions to weak boundary conditions and systems of hyperbolic conservation laws. The original wPINN framework prescribes Dirichlet boundary data for the solution on the entire boundary. However, prescribing Dirichlet boundary data is only possible on the (solution-dependent) inflow part of the boundary, so prescribing the correct boundary data requires prior knowledge of the solution. Secondly, the loss functional of original wPINNs is based on the Kruzhkov entropies, which is convenient because this formulation ensures one gets the unique weak entropy solution. However, the Kruzhkov entropies are only available for scalar hyperbolic conservation laws.

In light of this, our third goal is to extend wPINNs to incorporate weak boundary data understood in the sense of [24]. This approach is valid for scalar conservation laws in multiple space dimensions. We replace the Dirichlet-based boundary contribution of the loss by a suitable entropy-based inequality on the boundary of the domain.

During preliminary numerical experiments using wPINNs with weak boundary conditions we observe that long time domains may lead to poor approximate solutions. Sometimes, wPINNs choose to make a large error at early times, in order to remove discontinuities from the domain and then minimize residuals for a smooth function subsequently. While leading to overall worse solutions, this may seem advantageous because the loss is only minimized locally in the neural network's parameter space. This is inherently tied to the fact that PINNs do not respect causality. We present a time-slicing approach that enforces causal relationships between the time slices, but not within a single time slice. This resolves the encountered difficulties.

Our last goal is to extend wPINNs to systems of hyperbolic conservation laws. This means replacing the Kruzhkov entropy residual in the original wPINN loss by the PDE residual measured using a suitable weak norm. We do this in the scalar case already, as it seems to speed up training. Additionally, one has to ensure that one obtains the unique entropy solution. We fix a single strictly convex entropy-entropy flux pair and define an entropy residual that measures the violation of the corresponding entropy inequality. This is justified, because for scalar conservation laws with convex flux, satisfying a single entropy condition is equivalent to satisfying all entropy conditions. Further, for systems, there often is only a single physically motivated entropy. Our modified loss then naturally extends to systems of hyperbolic conservation laws by summing over the weak norms of the PDE residual of each individual equation.

Our manuscript is structured as follows. In Section 2 we give some background on various related machine learning methods used for solving PDEs. In Section 3 we provide new insights into why classical PINNs fail for discontinuous solutions of nonlinear hyperbolic conservation laws and why wPINNs are a suitable framework for their treatment. Then, Section 4 presents a novel approach for approximating weak norms using neural networks, in order to make wPINNs more efficient. Further, we extend wPINNs to weak boundary conditions for scalar conservation laws and extend them to systems of hyperbolic conservation laws. In Section 5 we describe our training algorithm and then show numerical experiments comparing the performance of wPINNs as described in [7] against our modified version, using the same examples as in [7]. For some basic examples, original wPINNs and our modified wPINNs give nearly identical results. However, in the more challenging tests, the modified wPINNs clearly outperform the original wPINNs. Finally, in Section 6, we demonstrate the effectiveness of our extensions to weak boundary conditions and systems on some examples. In Appendix A we provide an additional, more general, computation explaining the failure of PINNs. In Appendix B we give further details on the implementation of our modified wPINNs in order to make results more transparent and reproducible.

2. Background on PINNs

Originally, PINNs go back to [10, 25, 26], but were popularized more recently in [37]. They are an unsupervised learning method, not requiring labeled data from observations or numerical schemes. However, when data is available it can be readily incorporated. Interest in PINNs has grown rapidly due to their versatility. One of their main benefits is their meshless nature, which may make them suitable for high-dimensional problems. Some recent examples of these methods include [40], introducing the Deep Ritz Method, a PINN framework for elliptic problems, [28], presenting advancements in enforcing essential boundary conditions for the aforementioned framework, [34], wherein PINNs are applied to fractional advection-diffusion equations and [19], proposing a domain decomposition framework for PINNs for multiscale problems and to improve parallelization during training.

In their most frequently used form, PINNs are based on minimizing the L^2 norm of the residual associated with the PDE, admitting interpretation of PINNs as a weighted least-squares point-collocation method [2, 36]. There has been tremendous progress in the convergence analysis of PINNs [6, 32, 39], that is based on three key principles [7]:

(1) *Regularity* of the solutions to the underlying PDEs, (. . .) which can be leveraged into proving that PDE residuals can be made arbitrarily small.

(2) *Coercivity* (or stability) of the underlying PDEs, which ensures that the total error may be estimated in terms of residuals. For nonlinear PDEs, the constants in these coercivity estimates often depend on the regularity of the underlying solutions.

(3) *Quadrature error* bounds for estimating the so-called generalization gap between the continuous and discrete versions of the PDE residual [5].

As mentioned in [7], this analysis is not applicable to weak solutions, because of their reduced regularity. In [33], it is indeed observed in numerical experiments that classical PINNs fail for discontinuous solutions to nonlinear hyperbolic conservation laws. We explain this in Section 3. Because PINNs can be understood as a point-collocation method, they share some basic ideas with finite-element based point-collocation methods. For these, it has been observed that it is crucial which norm is used for minimization. In [16], it is shown that for advection-reaction problems with ill-posed boundary conditions, L^2 minimization of PDE residuals does not recover the unique viscosity solution in the sense of [1] to the problem. Instead, [17, 18] show that for linear first-order PDEs in one dimension with ill-posed boundary conditions, minimizing residuals with respect to the L^1 norm in the space of piecewise linear functions for given boundary data is a fast and effective strategy that recovers the viscosity solution. In [17], the L^1 minimization technique is then extended to (some) non-linear PDEs in one dimension, such as the Hamilton–Jacobi equations. This is shown to be successful when the exact solution is continuous. However, [17] provides an example where L^1 minimization fails to recover the correct discontinuous viscosity solution for the inviscid Burgers equation.

An established strategy to approach this problem is viscous regularization [27]. Viscous regularization has been applied to PINNs too [37]. However, this raises the question how much viscosity one needs to use to obtain solutions that can be learned effectively, yet are still close to the original solution.

The choice of norm is important for PINNs as well and there has been recent research into suitable choices. In [45], it is suggested that for high-dimensional Hamilton–Jacobi–Bellmann equations, one should choose an L^p-based loss with sufficiently large p, and a procedure for training with respect to an L^∞-based loss is outlined. Approximating solutions with reduced regularity has led to multiple variants of the PINN methodology, such as non-diffusive neural networks (NDNN) [30], control volume PINNs [36], variational PINNs [22], the Deep Ritz Method [11] and weak PINNs [7].

Fitting with our later arguments in Section 3, NDNNs [30] do not use a single, continuous neural network to approximate discontinuous solutions. Instead, discontinuity lines separating subdomains with smooth solutions are learned from the Rankine–Hugoniot jump conditions and a separate network is trained on each smooth subdomain. However, for a solution with d discontinuities, this requires a total of $2d + 1$ neural networks to express the approximate solution. For one-dimensional problems, this approach works well, but it is unclear how to extend this method into higher dimensions.

For control volume PINNs, the conservation law is expressed in integral form over test volumes, giving a formulation reminiscent of finite volume methods. The underlying idea of the other methods can be understood as minimizing the residual not in the L^2 norm, but in some weak (dual) norm. While the Deep Ritz Method is specifically tailored to elliptic problems by using the fact that their solutions are minimizers of energy functionals, variational PINNs and weak PINNs address more general PDEs. Both of these methods measure the residual in a dual norm, but the key difference between the two is the way in which the test functions are represented. While variational PINNs use a mesh-based approach to represent test functions, weak PINNs represent the test functions by neural networks, thereby retaining the meshless nature of the method. The concept of weak PINNs was introduced on the prototypical example of scalar hyperbolic conservation laws in [7]. The authors provide a detailed convergence analysis as well as numerical experiments demonstrating the capability to approximate discontinuous solutions. Another approach to solving problems with discontinuous solutions is outlined in [29] or [12], based on minimizing the residual in a weighted L^2 norm. The weight function depends on the gradient of the approximate solution and is chosen such that residuals close to steep gradients are weighted less.

While our wPINNs are focused on hyperbolic conservation laws with a (strictly) convex flux function, there have also been developments for conservation laws with non-convex fluxes. In [8] and [14] the authors show that classical PINNs cannot capture shock formation for the Buckley–Leverett problem. By replacing the flux function with its convex hull, they obtain the correct entropic shock in numerical experiments.

3. Explaining the failure of classical PINNs

In this section, we present our analytical arguments that explain why classical PINNs cannot approximate discontinuous solutions to nonlinear hyperbolic conservation laws well. We begin by recalling some basic definitions involving scalar conservation laws and classical PINNs before showing that continuous approximations of discontinuous solutions have large L^p norms of PDE residuals and thus will not be learned through minimizing the neural network loss. Then we motivate that weak norms do not have this issue.

3.1. Scalar Conservation Laws

We consider the scalar conservation law given by

$$\begin{cases} u_t + f(u)_x = 0 & \text{in } [0,\mathrm{T})\times\mathbb{R}, \quad (3.1)\\ \quad u(0,\cdot\,) = u_0 & \text{on } \mathbb{R}, \quad (3.2)\end{cases}$$

where u is the *conserved quantity*, $f \in C^2(\mathbb{R})$ is the *flux function* and $u_0 \in L^\infty(\mathbb{R})$ is the initial data. In the following, we always assume that $f \in C^2(\mathbb{R})$ and f is strictly convex.

Oftentimes one wishes to work not on all of $[0,\mathrm{T})\times\mathbb{R}$, but rather restrict oneself to a compact interval $D \subset \mathbb{R}$ in space. In this case one must supply boundary conditions, which need to be understood in a suitable weak sense, see [1, 24] and references therein. However, for initial conditions that are constant everywhere except on some compact interval in space, and boundary conditions that are constant in time, one knows that waves propagate with some maximum speed, such that the solution is constant outside some trapezoidal region in space-time. For now, we limit our discussion to Dirichlet boundary conditions on space intervals D such that the left- and right boundary are outside of this trapezoid for the given time interval $[0,\mathrm{T})$. We denote the boundary conditions by $u(t,x) = g(t,x)$ for $(t,x) \in (0,T) \times \partial D$, with g satisfying the compatibility condition $u_0(x) = g(0,x)$ for $x \in \partial D$. Later, we extend wPINNs to weak boundary conditions.

Even for very simple flux functions and smooth initial conditions, one may obtain discontinuous solutions after finite time. Hence, one has to work with weak solutions:

Definition 3.1 (Weak Solution)**.** We call $u \in L^\infty([0,T)\times\mathbb{R})$ a weak solution of problem (3.1)–(3.2) for some initial datum $u_0 \in L^\infty(\mathbb{R})$, if it satisfies

$$\int_{[0,T)}\int_{\mathbb{R}} u\varphi_t + f(u)\varphi_x \,\mathrm{d}x\,\mathrm{d}t + \int_{\mathbb{R}} u_0\varphi(0,\cdot)\,\mathrm{d}x = 0, \tag{3.3}$$

for any test function $\varphi \in C_c^1([0,T)\times\mathbb{R})$.

Weak solutions are, in general, not unique. One strategy to establish uniqueness is by imposing *entropy admissibility conditions* on weak solutions [3]. We define entropies η and corresponding entropy fluxes q as follows.

Definition 3.2 (Entropy – Entropy Flux Pair)**.** Let $\eta, q \in C^1(\mathbb{R})$. We say that (η, q) are a *convex entropy-entropy flux pair*, if η is strictly convex and $q' = f'\eta'$ almost everywhere.

Using the notion of entropy-entropy flux pairs one may define *entropy-admissible* solutions to problem (3.1)–(3.2).

Definition 3.3 (Entropy Admissibility)**.** We call u an *entropy-admissible* solution to the Cauchy problem (3.1)–(3.2) if it is a weak solution in the sense of Definition 3.1 and for any convex entropy-entropy flux pair, it satisfies

$$\eta(u)_t + q(u)_x \leq 0 \quad \text{in } [0,\mathrm{T})\times\mathbb{R} \tag{3.4}$$

in a distributional sense.

For a strictly convex flux function f, weak solutions u satisfying equation (3.4) for a single strictly convex entropy-entropy flux pair are entropy-admissible [4, 35]. Further, entropy-admissible solutions are unique.

3.2. Classical PINNs

In the following, we summarize the classical PINNs approach. We discuss fully connected feed-forward (FCFF) neural networks with hidden layers of uniform width to approximate solutions to scalar conservation laws. Note that there are a multitude of more sophisticated network architectures, as studied in other works, e.g. [11, 40]. Mathematically we define such neural networks as follows:

Definition 3.4 (Fully Connected NN)**.** Let $l, w \in \mathbb{N}$. Let $d \in \mathbb{N}$ be the input dimension. Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a smooth, non-linear function. We call σ the *activation function.*

Set $L^{[0]}(x) := W^{[0]}x + b^{[0]}$ with $W^{[0]} \in \mathbb{R}^{w\times d}$ and $b^{[0]} \in \mathbb{R}^{w}$. Further, set $L^{[i]}(x) := W^{[i]}x + b^{[i]}$ with $W^{[i]} \in \mathbb{R}^{w\times w}$ and $b^{[i]} \in \mathbb{R}^{w}$ for $i = 1, \dots, l-1$. Finally, set $L^{[l]}(x) := W^{[l]}x + b^{[l]}$ with $W^{[l]} \in \mathbb{R}^{1\times w}$ and $b^{[l]} \in \mathbb{R}$. We call the matrices $W^{[i]}, i = 0, \dots, l$ the *weights* and $b^{[i]}$ the *biases* of a neural network. The affine linear maps $L^{[i]}$ are called the *layers* of the neural network. We say that the network has l layers with width w, where layers $1, \dots, l-1$ are called *hidden layers.*

The set of weights and biases is called the *parameters* of the neural network, given by $\theta := \{W^{[i]}, b^{[i]}\}_{i=0,\dots,l}$.

Then, the *neural network* associated with the parameters θ is a map $u_\theta \in C^\infty(\mathbb{R}^d, \mathbb{R})$ given by

$$u_\theta = L^{[l]} \circ \sigma \circ L^{[l-1]} \circ \sigma \dots \circ \sigma \circ L^{[1]} \circ \sigma \circ L^{[0]}, \tag{3.5}$$

where we understand the composition with σ to mean the element-wise application to the layer output.

We limit our analysis to FCFF networks to focus on the effect of the choice of loss function, however our arguments extend to different architectures as well. Further, we limit ourselves to smooth activation functions. It should be noted that a popular choice of activation is the $\mathrm{ReLU}(\,\cdot\,) := \max(0, \cdot)$ function, which is not smooth. Our analysis also applies to ReLU networks.

PINN-based methods are based on minimizing the PDE residuals measured in some discrete approximation of an L^p norm, most commonly for $p = 2$.

For $v \in C^1([0, \mathrm{T}) \times D)$, we define the pointwise PDE residual as

$$\mathcal{R}[v] := v_t + f(v)_x \in C^0([0, \mathrm{T}) \times D). \tag{3.6}$$

Let u_θ be a neural network with parameters θ. Then the classical PINN approach defines the interior contribution to the neural network loss function as a Monte-Carlo approximation of the L^2-in-space-time norm of $\mathcal{R}[u_\theta]$. This is done by randomly choosing *collocation points* $\{(t_i, x_i)\}_{i=1}^{N_c}$ with $t_i \in [0, \mathrm{T})$, $x_i \in D$ and N_c the number of collocation points, and then computing

$$\mathfrak{L}_{\mathrm{int}}(u_\theta) := \frac{1}{N_c}\sum_{i=1}^{N_c} |\mathcal{R}[u_\theta](t_i, x_i)|^2. \tag{3.7}$$

Initial- and boundary data are imposed using further penalty terms, given by

$$\mathfrak{L}_{\mathrm{ic}}(u_\theta) = \|u_\theta(0, \cdot\,) - u_0\|_{2,\mathrm{MC}}^2 \qquad \text{and} \qquad \mathfrak{L}_{\mathrm{bc}}(u_\theta) = \|u_\theta - g\|_{2,\mathrm{MC}}^2, \tag{3.8}$$

where $\|\cdot\|_{2,\mathrm{MC}}$ denotes a Monte-Carlo approximation to the L^2-norm on D for the initial data and on $(0, T) \times \partial D$ for the boundary data. The overall loss is then the weighted sum of interior, initial and boundary loss, $\mathfrak{L}(u_\theta) := \mathfrak{L}_{\mathrm{int}} + \lambda\,(\mathfrak{L}_{\mathrm{ic}} + \mathfrak{L}_{\mathrm{bc}})$ with some user-chosen hyperparameter $\lambda > 0$ setting the relative importance of the two terms [48]. As empirically observed in [31], this approach based on minimizing the L^2 norm of the interior residual does not work for learning discontinuous solutions to nonlinear hyperbolic conservation laws.

3.3. Choice of Norm

In the following, we present an illustrative argument that shows why classical PINNs cannot learn such discontinuous solutions based on the example of standing shock solutions to the inviscid Burgers equation. The intuition we develop here easily extends to other nonlinear hyperbolic conservation laws and moving shocks and highlights a fundamental issue. While the computations are elementary, we could not find an argument of this type in the literature. Instead, the failure of PINNs was treated as an experimental observation so far.

Note that the failure of PINNs is a nonlinear effect and will not occur in the linear case. In fact, in the linear case we expect minimizing L^p norms of the PDE residual for $1 \leq p \leq \infty$ to work well. For linear advection with constant velocity, by learning a simple linear coordinate transformation, it is clear that the residual can be made to vanish exactly. It remains to learn initial- and boundary data. However, neural networks can easily learn approximations of discontinuous initial data, giving good approximate solutions for linear problems.

Consider that by construction, for continuous activation functions σ, neural networks are continuous as well, so they cannot represent discontinuities exactly. As such, one expects a reasonable neural network approximation to a discontinuous solution to learn smoothed-out versions of shocks, such that it will be close to the exact solution in a pointwise sense outside of a small region around discontinuities and in an L^2-sense for the entire domain.

As a concrete example we consider the inviscid Burgers equation, that is the scalar conservation law for the flux function $f(u) = \frac{1}{2}u^2$ and the initial condition

$$u_0(x) = \begin{cases} 1 & \text{for } x < 0, \\ -1 & \text{for } x \geq 0, \end{cases} \tag{3.9}$$

having a standing shock as its entropy solution.

As a first scenario we consider time-independent approximations to the solution of this problem, as the exact solution is time-independent as well and this avoids technical complications. We consider approximate solutions $\tilde{u}$ which are smooth and satisfy

$$\begin{cases} \tilde{u}(x) \in [1-\epsilon, 1+\epsilon] & \text{for } x < -\epsilon, \\ \tilde{u}(x) \in [-1-\epsilon, -1+\epsilon] & \text{for } x > \epsilon, \\ \tilde{u}(x) \in [-1-\epsilon, 1+\epsilon] & \text{for } -\epsilon \leq x \leq \epsilon \end{cases} \tag{3.10}$$

for some $\epsilon > 0$. For $\epsilon \to 0$ one easily checks that such approximations converge towards the exact solution in $L^2([0,\mathrm{T}) \times D)$ with a rate of $\sqrt{\epsilon}$. Our key argument is that these reasonable approximations will, in fact, not produce a small L^2 norm of the PDE residual, such that PINNs would not learn such approximations during the optimization process. We show that the L^2 norm of the residual scales as $\frac{1}{\sqrt{\epsilon}}$ in this case, i.e. better shock resolution leads to residuals with larger L^2 norm.

We consider only the contribution to the residual from $[0,\mathrm{T}) \times (-\epsilon, \epsilon)$, as the shock is what causes the L^2 norm of the residual to grow large and

$$\|\mathcal{R}[\tilde{u}]\|_{L^2([0,\mathrm{T}) \times D)} \geq \|\mathcal{R}[\tilde{u}]\|_{L^2([0,\mathrm{T}) \times (-\epsilon,\epsilon))}.$$

Additionally, Hölder's inequality implies that $\|\mathcal{R}[\tilde{u}]\|_{L^1([0,\mathrm{T}) \times (-\epsilon,\epsilon))} \leq \sqrt{2\epsilon T}\, \|\mathcal{R}[\tilde{u}]\|_{L^2([0,\mathrm{T}) \times (-\epsilon,\epsilon))}$. Further, we note that for $\epsilon < 1$, because $\tilde{u}(-\epsilon) > 0$ and $\tilde{u}(\epsilon) < 0$ and $\tilde{u}$ is continuous, there is at least one zero $\bar{x}$ of $\tilde{u}$ in $(-\epsilon, \epsilon)$. Hence, we have $f(\tilde{u}(\bar{x})) = 0$. This behavior close to the discontinuity is what makes the L^2 norm of the residual large, because contrary to the true solution which satisfies

$f(u) = \frac{1}{2}$ everywhere, there is a region where $|f(\tilde{u})_x|$ is large. Then we compute:

$$\begin{aligned} \|\mathcal{R}[\tilde{u}]\|_{L^1([0,\mathrm{T})\times(-\epsilon,\epsilon))} &= \int_0^T \int_{-\epsilon}^{\epsilon} |f(\tilde{u})_x| \,\mathrm{d}x \,\mathrm{d}t \\ &\geq \int_0^T \left| \int_{-\epsilon}^{\bar{x}} f(\tilde{u})_x \,\mathrm{d}x \right| \mathrm{d}t + \left| \int_{\bar{x}}^{\epsilon} f(\tilde{u})_x \,\mathrm{d}x \right| \mathrm{d}t \\ &= \int_0^T \frac{1}{2}\tilde{u}(-\epsilon)^2 + \frac{1}{2}\tilde{u}(\epsilon)^2 \,\mathrm{d}t \geq T\,(1-\epsilon)^2\,, \end{aligned} \tag{3.11}$$

where in the last line we inserted the flux function for the Burgers equation explicitly.

Due to our preliminary considerations the L^2 norm of the residual may be estimated by:

$$\|\mathcal{R}[\tilde{u}]\|_{L^2([0,\mathrm{T})\times D)} \geq \|\mathcal{R}[\tilde{u}]\|_{L^2([0,\mathrm{T})\times(-\epsilon,\epsilon))} \geq \frac{\|\mathcal{R}[\tilde{u}]\|_{L^1([0,\mathrm{T})\times(-\epsilon,\epsilon)}}{\sqrt{2\epsilon T}} \geq \frac{\sqrt{T}(1-\epsilon)^2}{\sqrt{2\epsilon}}. \tag{3.12}$$

As this estimate shows, for time-independent reasonable approximations to shock solutions of the Burgers equation, reducing ϵ, which intuitively makes the approximation better, increases the L^2-norm of its residual.

We are not able to prove a similar estimate for the general case of time-dependent approximations satisfying (3.10) for each t. However, we consider two examples of typical time dependence, which cover a wide range of possible approximations and show that these do not produce a small L^2-norm of the residual either. The more technical example is located in Appendix A.

Now we consider approximations whose solution profile is shifted with some time-dependent velocity, but whose shape does not change. This means we consider time-dependent approximations of the form $\tilde{u}(x - s(t))$ with $\tilde{u}$ as in (3.10) for a smooth function s denoting the overall shift of the solution. Then the time derivative of such approximations is given by $\tilde{u}_t = -s'\tilde{u}_x$. The velocity of the profile is given by $v := s'$. As before, we compute

$$\begin{aligned} \int_0^T \int_{-\epsilon+s}^{\epsilon+s} |f(\tilde{u})_x + \tilde{u}_t| \,\mathrm{d}x \,\mathrm{d}t &\geq \int_0^T \left| \int_{-\epsilon+s}^{\bar{x}+s} f(\tilde{u})_x - v\tilde{u}_x \,\mathrm{d}x \right| \mathrm{d}t + \int_0^T \left| \int_{\bar{x}+s}^{\epsilon+s} f(\tilde{u})_x - v\tilde{u}_x \,\mathrm{d}x \right| \mathrm{d}t \\ &= \int_0^T \left| \underbrace{f\left(\tilde{u}(-\epsilon)\right) - v(t)\tilde{u}(-\epsilon)}_{=:A} \right| + \left| \underbrace{f\left(\tilde{u}(\epsilon)\right) - v(t)\tilde{u}(\epsilon)}_{=:B} \right| \mathrm{d}t \\ &\geq \frac{T}{2}\,(1-\epsilon)^2 + (1-\epsilon)\|v\|_{L^1([0,\mathrm{T}))}. \end{aligned} \tag{3.13}$$

For the last inequality, we use that $\tilde{u}(\epsilon)$ and $\tilde{u}(-\epsilon)$ have opposing sign. Then, regardless of the sign of $v(t)$, either A or B is the sum of two positive contributions, while the other has mixed sign. For the sum of two positive terms, we may leave out the absolute value and we drop the contribution from the mixed term. The same arguments as before show that the L^2 norm of the residual grows as $\frac{1}{\sqrt{\epsilon}}$ again.

We see that for fairly general reasonable approximations, the L^2-in-space-time norm of the PDE residual grows large for small ϵ, while the magnitude of the L^1 norm does not seem to be connected to ϵ. This shows that both these norms are unsuitable choices for the loss function of PINNs then, because training networks by minimizing the resulting loss functions will not learn approximations with small ϵ.

Instead, one should consider a weaker norm of the residual. We argue that using the $L^2(0,T;W^{-1,p}(D))$- norm (L^2-$W^{-1,p}$-norm for short) with $1 < p < \infty$ is a good choice. Consider again the toy problem as outlined in (3.9) with a smooth approximation $\tilde{u}$ as in (3.10). For technical simplicity we will limit ourselves to approximations that are constant in time, that is $\tilde{u}(t,x) = \tilde{u}(0,x)$ for all (t,x) in $[0,\mathrm{T})\times D$.

We show that the L^2-$W^{-1,p}$ norm of the residual of approximate solutions of the form (3.10) goes to zero for $\epsilon \to 0$ provided that the sequence is sufficiently non-oscillatory, i.e. if $\epsilon^{1+\frac{1}{p}}\|f(\tilde{u})_x\|_{L^\infty([0,\mathrm{T})\times(-\epsilon,\epsilon))}$ and $\|f(\tilde{u})_x\|_{L^2([0,\mathrm{T});L^p(D\setminus(-\epsilon,\epsilon)))}$ both go to zero, which is a desirable property of an approximation.

Fix $1 < p < \infty$ and let $\frac{1}{p} + \frac{1}{q} = 1$. Let $V \coloneqq W_0^{1,q}(D)$ with $\|\cdot\|_V \coloneqq \|\cdot\|_{W^{1,q}}$, and let $S \coloneqq \{\varphi \in V \colon \|\varphi\|_V = 1\}$. Then by definition of the $W^{-1,p}$ norm, we compute

$$\begin{aligned}\|\mathcal{R}[\tilde{u}]\|^2_{L^2([0,\mathrm{T});W^{-1,p}(D))} &= \int_{[0,\mathrm{T})}\left(\sup_{\varphi\in S}\int_D f(\tilde{u})_x\varphi\,\mathrm{d}x\right)^2\mathrm{d}t\\ &\le \int_{[0,\mathrm{T})}\left(\sup_{\varphi\in S}\int_{(-\epsilon,\epsilon)} f(\tilde{u})_x\varphi\,\mathrm{d}x\right)^2\mathrm{d}t + \int_{[0,\mathrm{T})}\left(\sup_{\varphi\in S}\int_{D\setminus(-\epsilon,\epsilon)} f(\tilde{u})_x\varphi\,\mathrm{d}x\right)^2\mathrm{d}t\end{aligned} \tag{3.14}$$

To estimate the first term of (3.14), we rewrite $\varphi(x) = \varphi(0) + \int_0^x \varphi'(s)\,\mathrm{d}s$ and use the fact that $|(f(\tilde{u}(t,\epsilon)) - f(\tilde{u}(t,-\epsilon))| \le 4\epsilon$ for the Burgers equation with approximations $\tilde{u}$ as in (3.10) to conclude that

$$\begin{aligned}\int_0^T\left(\sup_{\varphi\in S}\int_{-\epsilon}^{\epsilon} f(\tilde{u})_x\varphi\,\mathrm{d}x\right)^2\mathrm{d}t &= \int_0^T\left(\sup_{\varphi\in S}\Bigg(\varphi(0)\left(f(\tilde{u}(t,\epsilon)) - f(\tilde{u}(t,-\epsilon))\right)\right.\\ &\qquad\left.+\int_{-\epsilon}^{\epsilon} f(\tilde{u})_x\int_0^x \varphi'(s)\,\mathrm{d}s\,\mathrm{d}x\Bigg)\right)^2\mathrm{d}t\\ &\le \int_0^T\left(\sup_{\varphi\in S}\|f(\tilde{u})_x\|_\infty\int_{-\epsilon}^{\epsilon}\int_0^x|\varphi'(s)|\,\mathrm{d}s\,\mathrm{d}x + 4\epsilon\varphi(0)\right)^2\mathrm{d}t\\ &\le \int_0^T\left(\sup_{\varphi\in S}\|f(\tilde{u})_x\|_\infty\int_{-\epsilon}^{\epsilon}|x|^{\frac{1}{p}}\|\varphi'\|_{L^q(D)}\,\mathrm{d}x + 4\epsilon\varphi(0)\right)^2\mathrm{d}t\\ &= T\sup_{\varphi\in S}\left(\|f(\tilde{u})_x\|_\infty\left(\frac{2}{1+1/p}\epsilon^{1+\frac{1}{p}}\right)\|\varphi'\|_{L^q(D)} + 4\epsilon\varphi(0)\right)^2\\ &\le T\left(\|f(\tilde{u})_x\|_\infty\left(\frac{2}{1+1/p}\epsilon^{1+\frac{1}{p}}\right) + 4\epsilon c_S\right)^2.\end{aligned} \tag{3.15}$$

The last inequality uses that, by definition, $\|\varphi'\|_{L^q(D)} \le 1$ and that in one dimension $W^{1,q}$ with $q > 1$ is continuously embedded in L^∞, with embedding constant denoted by c_S.

Estimating the second term of (3.14) is straightforward. Note that by definition of φ and applying Hölder's inequality, one sees that

$$\int_{[0,\mathrm{T})}\left(\sup_{\varphi\in S}\int_{D\setminus(-\epsilon,\epsilon)} f(\tilde{u})_x\varphi\,\mathrm{d}x\right)^2\mathrm{d}t \le \int_{[0,\mathrm{T})}\left(\int_{D\setminus(-\epsilon,\epsilon)} f(\tilde{u})_x^p\,\mathrm{d}x\right)^{\frac{2}{p}}\mathrm{d}t = \|f(\tilde{u})_x\|^2_{L^2([0,\mathrm{T});L^p(D\setminus(-\epsilon,\epsilon)))}. \tag{3.16}$$

Combining the estimates for each of the two previous terms, we may estimate the residual in this norm by

$$\|\mathcal{R}[\tilde{u}]\|^2_{L^2([0,\mathrm{T});W^{-1,p}(D))} \le T\left(\|f(\tilde{u})_x\|_\infty\left(\frac{2}{1+1/p}\epsilon^{1+\frac{1}{p}}\right) + 4\epsilon c_S\right)^2 + \|f(\tilde{u})_x\|^2_{L^2([0,\mathrm{T});L^p(D\setminus(-\epsilon,\epsilon)))}. \tag{3.17}$$

The last term of estimate (3.17) is not connected to the resolution of the shock. Comparing the two terms inside the square, because $\|f(\tilde{u})_x\|_\infty \ge \frac{(1-\epsilon)^2}{2\epsilon}$, for small ϵ, the first term scales with $\epsilon^{\frac{1}{p}}$ while the second term scales with ϵ, so the first term is the dominant contribution for small ϵ. Larger values of p produce steeper gradients of the loss with respect to ϵ when ϵ is small and will incentivize finer shock resolution. However, choosing larger p will not always be better, because larger p have smaller gradients for large values of ϵ, which occur at the beginning of training neural networks. Thus, it is unclear what the best choice of p is. In our experience, $p = 2$ has worked very well and leads to the simplest setup.

Overall, the estimate shows that the L^2-$W^{-1,p}$ norm with $1 < p < \infty$ is a good choice of norm for the PDE residual in the neural network loss function, because good approximations of the exact solution correspond to small losses. Approximations oscillating in $[0, \mathrm{T}) \times (-\epsilon, \epsilon)$ will lead to larger values of $\|f(\tilde{u})_x\|_\infty$. Such approximations do not guarantee a small norm of the residual, showing that the goal functional discourages spurious oscillations.

Remark 3.5. The above arguments were outlined in the case of a stationary shock, but can be adapted to cover moving shocks. The estimates employed are done pointwise for each $t \in [0, \mathrm{T})$, such that the only difference is that the interval $(-\epsilon, \epsilon)$ now moves with the proper shock velocity instead of being fixed.

The original wPINNs fit into the previous analysis. They introduce a *(Kruzhkov) entropy residual* [7], based on enforcing entropy inequalities, which is then turned into a loss function by measuring it in something similar to the $L^2(0, T; W^{-1,2}(D))$ norm. For $\bar{S} := \{\varphi \in C_c^\infty([0, \mathrm{T}) \times D) : \varphi \geq 0, \|\partial_x \varphi\|_{L^2([0,\mathrm{T}) \times D)} = 1\}$, the loss function on the continuous level is given by

$$\mathcal{L}^K_{\text{ent}}(u_\theta) = \sup_{\varphi \in \bar{S}} \max_{c \in \mathbb{R}} \int_{[0,\mathrm{T})} \int_D \varphi \partial_t |u_\theta - c| - \operatorname{sign}(u_\theta - c)(f(u_\theta) - f(c))\varphi_x \,\mathrm{d}x \,\mathrm{d}t \,. \tag{3.18}$$

Note that this form of loss contains a nested maximization problem over $c \in \mathbb{R}$, which is approximated in [7] by sampling some discrete set of c and computing the maximum directly.

In contrast to our previous arguments, $\mathcal{L}^K_{\text{ent}}$ is not based on computing a norm of the PDE residual directly, however the Kruzhkov entropy inequalities already encode the PDE itself. Being based on entropy conditions, this choice of loss ensures that one finds the unique entropy solution.

4. Modified wPINNs and Extensions

4.1. Weak Norm Estimation

Our previous arguments show that classical PINNs cannot succeed for hyperbolic conservation laws and it is instead required to use PINNs based on dual norms for this problem class. We use a novel approach for computing dual norms using neural networks by solving a dual elliptic problem. This improves the computational efficiency of weak PINNs by giving better approximations of the dual norm with less training. The modified loss functional accelerates learning in terms of the number of epochs the neural networks require to be trained to comparable accuracy, or achieve higher accuracy after equal number of epochs.

To fix notation, we first describe the loss components on an analytical level. We replace the Kruzhkov entropy formulation from the original wPINNs by the PDE residual and an entropy residual based on a single strictly convex entropy, because systems of hyperbolic conservation laws commonly only have one physically motivated entropy. Note that this is independent of our approach to computing dual norms, which can be applied to the Kruzhkov entropy formulation of the original wPINNs as well.

The interior PDE residual loss measures the PDE residual in the L^2-$W^{-1,p}$ norm. Recall that $S = \{\varphi \in V \colon \|\varphi\|_V = 1\}$ with $V = W^{1,q}_0(D)$ and q is the Hölder-conjugate exponent of p. The interior PDE residual loss is then given by

$$\mathcal{L}^u_{\text{int}}(u_\theta) = \int_{[0,\mathrm{T})} \left(\sup_{\varphi \in S} \int_D (\partial_t u_\theta + \partial_x f(u_\theta))\, \varphi \,\mathrm{d}x \right)^2 \mathrm{d}t \,. \tag{4.1}$$

This loss is minimized by any weak solution, even ones which are not entropy solutions, so enforcing an entropy inequality is required. In [7], a numerical example approximating a non-entropic weak solution is shown. Provided f is strictly convex, a single strictly convex entropy-entropy flux pair is sufficient to enforce entropy admissibility, instead of the family of Kruzhkov entropies. Thus, we fix one such entropy-entropy flux pair (η, q) and introduce the *entropy loss*, measured in the L^2-$W^{-1,p}$ norm, as

$$\mathcal{L}^{\eta}_{\mathrm{ent}}(u_\theta) = \int_{[0,\mathrm{T})} \left(\sup_{\xi\in S} \int_D (\eta(u_\theta)_t + q(u_\theta)_x)^{\oplus}\, \xi \,\mathrm{d}x \right)^2 \mathrm{d}t\,, \tag{4.2}$$

where we take the positive part denoted by $(\,\cdot\,)^{\oplus}$ since the entropy condition is an inequality. This approach avoids the nested maximization over $c \in \mathbb{R}$ of the original wPINN approach in favor of a second, independent, maximization problem with another adversarial neural network that one trains using gradient ascent.

We now discuss our strategy for approximating the $W^{-1,p}$ norm, whose definition takes a supremum over all $\varphi \in S$. When parameterizing approximations to φ using a neural network, it is unclear how to enforce the normalization of the test function through the network design, so one has to compute the $W^{1,q}$ norm (using Monte-Carlo sampling) and divide by this. We believe that the division by such a norm makes the maximization problem more difficult. Also, by construction, this formulation is invariant under scalar rescaling of the network approximating the dual norm. As the updates during gradient ascent change the normalization of the network, this undetermined degree of freedom may reduce the speed and accuracy of the norm estimation.

To remedy these downsides, we propose computing the $W^{-1,p}$ norm by learning the solution to a q-Laplace problem with $\frac{1}{p} + \frac{1}{q} = 1$. We identify the dual space $W^{-1,p}(D) = (W^{1,q}_0(D))'$. Given a functional $v \in W^{-1,p}(D)$ for $1 < p < \infty$, we see that determining

$$w \in W^{1,q}_0(D) \text{ with } \int_D |\nabla w|^{q-2} \nabla w \nabla \zeta \,\mathrm{d}x = \int_D v\zeta \,\mathrm{d}x \quad \text{for all } \zeta \in W^{1,q}_0(D), \tag{4.3}$$

one has $\|v\|^p_{W^{-1,p}(D)} = |w|^q_{W^{1,q}(D)} := \|\nabla w\|^q_{L^q(D)}$. The norm relationships between v and w are derived in [9, Theorems 7 and 8]. Note that due to the zero-boundary conditions, by Poincaré inequality the $W^{1,q}$-seminorm is a proper norm on $W^{1,q}_0(D)$. The solution of this dual problem may also be characterized as the maximizer of the energy functional

$$I(w) = \int_D vw \,\mathrm{d}x - \frac{1}{q}\int_D |\nabla w|^q \,\mathrm{d}x\,. \tag{4.4}$$

The equivalence of these two characterizations is given in [9, equation (12)], and the explanations thereafter. We use this characterization to train our test functions, where the PDE residual $\mathcal{R}[u_\theta]$ takes the role of v and gradient ascent with respect to w is performed to approximate the test function. In the case $q = 2$, maximizing $I(w)$ on the set of neural functions is known as the "Deep Ritz Method" [11]. It is effective even for low-regularity right-hand-sides v for the Poisson problem [20].

4.2. Loss Definition

Next, we discretize the continuous loss functional using Monte-Carlo sampling. We introduce interior collocation points $S_{\mathrm{int}} := \{(t_i, x_i)\}_{i=1}^{N_{\mathrm{int}}}$ with $t_i \in [0, \mathrm{T})$ and $x_i \in D$ and N_{int} the number of collocation points. Similarly, we have S_{ic} sampling D for the initial data and S_{bc} sampling $(0, T) \times \partial D$ for the boundary data.

We work with three neural networks u_θ, $\tilde{\varphi}_\chi$ and $\tilde{\xi}_\nu$ with weights denoted by θ, χ and ν. The latter two networks approximate the solutions to the dual problems as above, so one needs to enforce zero boundary conditions through either soft or hard constraints [20]. Soft constraints add an additional loss term penalizing the violation of boundary conditions. Hard constraints encode the boundary condition exactly into the network design. We opt for hard constraints, because this speeds up learning and improves accuracy of PINNs [20]. We enforce the boundary conditions through a cutoff function $w : D \to \mathbb{R}^+$ with $w|_{\partial D} = 0$, $w'|_{\partial D} \neq 0$ and $w(x) > 0$ for all x in the interior of D. Then the functions $\varphi_\chi(x,t) := \tilde{\varphi}_\chi(x,t)w(x)$ and $\xi_\nu(x,t) := \tilde{\xi}_\nu(x,t)w(x)$ satisfy zero boundary conditions.

To simplify notation we define

$$\mathscr{r}_{\text{PDE}}(u_\theta; \varphi_\chi) := (\partial_t u_\theta)\varphi_\chi - f(u_\theta)(\partial_x \varphi_\chi). \tag{4.5}$$

This definition corresponds to the integrand of the PDE residual loss after integration by parts in space, as the boundary contributions vanish because the network φ_χ has zero boundary conditions in space. The integration by parts when defining the loss for numerical simulations is also done in [7] and gives good results, so we adopt this approach. The overall PDE residual loss is then discretized as

$$\mathfrak{L}^u_{\text{PDE}} := \frac{1}{N_{\text{int}}}\left(\sum_{S_{\text{int}}} \mathscr{r}_{\text{PDE}}(u_\theta;\varphi_\chi)(t_i,x_i) - \frac{1}{q}|\nabla_x \varphi_\chi|^q(t_i,x_i)\right). \tag{4.6}$$

We solve the elliptic problem in space using a single network with the time t as an additional input parameter, such that it learns the solution for the dual problem simultaneously on the entire time interval. This is equivalent to the continuous definition of the loss, but more effective on the discrete level.

Note that, when φ_χ (approximately) solves the aforementioned dual elliptic problem with the PDE residual as right-hand side, (4.6) corresponds to the Monte-Carlo approximation of $\int_{[0,T)}\int_D \mathcal{R}[u_\theta]\varphi_\chi - \frac{1}{q}|\nabla_x\varphi_\chi|^q \,\mathrm{d}x\,\mathrm{d}t = \frac{1}{p}\int_{[0,\mathrm{T})}\int_D |\nabla_x\varphi_\chi|^q \,\mathrm{d}x\,\mathrm{d}t$. We compute the approximation of the L^2-$W^{-1,p}$ norm using equation (4.6) because the dependence on the parameters θ of the network u_θ is explicit then, as required for automatic differentiation.

Further, we have the *entropy residual* for a given entropy-entropy flux pair (η, q) defined as

$$\mathscr{r}_{\text{ent}}(u_\theta;\xi_\nu;(\eta,q)) := (\eta(u_\theta)_t + q(u_\theta)_x)^{\oplus}\,\xi_\nu, \tag{4.7}$$

which we measure again in the squared L^2-$W^{-1,p}$ norm as

$$\mathfrak{L}^\eta_{\text{ent}} := \frac{1}{N_{\text{int}}}\left(\sum_{S_{\text{int}}} \mathscr{r}_{\text{ent}}(u_\theta;\xi_\nu)(t_i,x_i) - \frac{1}{q}|\nabla_x \xi_\nu|^q(t_i,x_i)\right). \tag{4.8}$$

Putting these parts together we consider the following interior loss function:

$$\mathfrak{L}_{\text{int}} = \mathfrak{L}^\eta_{\text{ent}} + \mathfrak{L}^u_{\text{PDE}} \tag{4.9}$$

Sometimes, it might be possible to speed up training by weighting the PDE and entropy contribution relative to each other as well, however we leave out this weight in the following for simplicity. Note that the interior loss consists of two additive contributions, the PDE residual loss (4.6) and the entropy loss (4.8), each of which again consist of two parts. The PDE residual loss $\mathfrak{L}^u_{\text{PDE}}$ has no dependence on the adversarial network ξ_ν, and the spatial gradient part depends only on the network φ_χ, while the entropy loss $\mathfrak{L}^\eta_{\text{ent}}$ does not depend on φ_χ and its spatial gradient part depends only on ξ_ν.

For the overall loss we add in initial- and boundary contributions, for Dirichlet boundary data g on the spatial boundary of the domain and initial data u_0 at $t = 0$. This gives the overall loss function

$$\begin{aligned}\mathfrak{L}(u_\theta,\varphi_\chi,\xi_\nu) &= \mathfrak{L}_{\text{int}}(u_\theta,\varphi_\chi,\xi_\nu)\\ &\quad + \lambda\left(\frac{1}{N_{\text{ic}}}\sum_{S_{\text{ic}}}(u_\theta(0,x_i) - u_0(x_i))^2 + \frac{1}{N_{\text{bc}}}\sum_{S_{\text{bc}}}(u_\theta(t_i,x_i) - g(t_i,x_i))^2\right),\end{aligned} \tag{4.10}$$

with parameter $\lambda > 0$. These additional terms depend only on the solution network u_θ. The decomposition of the loss into additive terms independent of some of the networks allows avoiding some computations to improve efficiency. For example, computing a gradient of the loss with respect to only the parameters θ does not require computation of the spatial gradients of the networks φ_χ and ξ_ν in (4.6) and (4.8). The parameter λ may be chosen to balance gradients of the interior and boundary contributions during training [47].

To summarize, our numerical scheme is given by

$$(u_\theta, \varphi_\chi, \xi_\nu) = \underset{u_\theta}{\operatorname{argmin}} \left(\underset{\varphi_\chi, \xi_\nu}{\operatorname{argmax}} \, \mathfrak{L}(u_\theta, \varphi_\chi, \xi_\nu) \right), \tag{4.11}$$

where the minimization and maximization are performed over the respective sets of neural functions determined by their user-prescribed neural network architecture.

4.3. **Extension to Weak Boundary Conditions**

In this section, we show how to incorporate Dirichlet boundary conditions weakly in the sense of Bardos, Leroux and Nedelec [1] for scalar conservation laws, i.e. the value of g is only attained, if it lies on the inflow part of the boundary, while on the outflow part of the boundary, the boundary data need not be attained. We use the framework from [24]. Let u be a weak solution of (3.1)–(3.2) in the sense of (3.3).

By considering the zero-diffusion limit corresponding to the hyperbolic conservation law, [24] shows that solutions satisfying the bounded boundary data $u(t,x) = g(t,x)$ on $(0,T) \times \partial D$ weakly satisfy an entropy inequality on the boundary of the domain. It is given by

$$\int_{[0,\mathrm{T})} \int_{\partial D} \Big((q(g) - q(u)) \cdot n + \eta'(g) \, (f(u) - f(g)) \cdot n \Big) \varphi \, \mathrm{d}x \, \mathrm{d}t \le 0, \tag{4.12}$$

for all smooth $\varphi \ge 0$, where (η, q) are a fixed convex entropy-entropy-flux pair and n is the outer unit normal on ∂D. In one dimension, the inner integral can be simplified to a simple sum. However, we think equation (4.12) better shows the underlying structure of this approach and how to generalize it for multi-dimensional problems.

We wish to weakly enforce (4.12) with our wPINNs, replacing the standard PINNs boundary term $\|u_\theta - g\|^2_{L^2((0,T)\times\partial D)}$. While equation (4.12) contains a test function, crucially it does not contain any derivatives of the solution u or test function φ. Because of this, we may use a strong norm, like the L^2-norm, on the postive part of the boundary inequality to weakly enforce it. This gives the following loss contribution on the boundary of the domain:

$$\mathcal{L}_{\mathrm{bc}}(u_\theta) = \left\| \Big((q(g) - q(u_\theta)) \cdot n + \eta'(g) \, (f(u_\theta) - f(g)) \cdot n \Big)^{\oplus} \right\|^2_{L^2((0,T)\times\partial D)} \tag{4.13}$$

The discretization of this term proceeds as previously, by approximating the L^2-norm using Monte-Carlo integration on $(0,T) \times \partial D$.

4.4. **Causality**

Solutions to hyperbolic conservation laws have causal structure, meaning that information only propagates forward in time, with a limited speed. In contrast, PINNs do not enforce this structure. During numerical experiments for weak boundary conditions, we observed shocks being moved outside of the computational domain at small values of t instead of propagating at the correct speed.

We hypothesize that this is related to the lack of causality of PINNs. Shocks contribute the majority of the neural network loss, while the magnitude of the residual is significantly smaller away from shocks. On longer time domains, because gradient descent locally minimizes the loss in the parameters of the network, incurring a large residual error at early times t by removing shocks from the domain and having a smooth approximate solution on the rest of the domain may appear a promising descent direction. However, this leads to completely incorrect approximations due to getting stuck in a local loss minimum. This is due to PINNs not enforcing causality. If causality were enforced, making a large error at early times would no longer be "encouraged" through a potential reduction in loss at later times.

Note that this problem may occur with strong boundary data enforced using a penalty as well, when the weight of the boundary loss is too small. Increasing the weight makes violating the boundary conditions prohibitively expensive and can somewhat mitigate this behavior. However, the option of committing a large error at early times and then learning some unrelated smooth solution that is compatible with the boundary data still remains. With weak boundary conditions, changing the weighting of the loss contributions is ineffective. Weak boundary conditions do not penalize the network for moving the shock from the domain, so increasing their weight does not change the situation. Only the residual loss penalizes the network for making a large error at early times, and weighting the residual more also increases projected payoff of having a smooth solution after making a large error.

To enforce some level of causality, we subdivide the time domain into smaller slices and subsequently train neural networks on each time slice. Details can be found in Section 6.1 where we apply our time-slicing approach. This approach enforces causality between each of the time slices, while on each time slice there is no causality. Another possible approach would be "causal training" [46], that continuously discounts the loss at later times t based on the size of loss at earlier times, discouraging large errors for small t. However, we did not try this alternative.

5. Numerical Results - Performance Comparison

In this section we present several numerical experiments to compare the performance of the modified wPINNs against the original wPINNs. We train neural networks for initial data corresponding to a standing shock, a moving shock, a rarefaction wave from Riemann problem initial data and sine initial data, to match the range of examples covered in [7].

5.1. Algorithm and Implementation Details

First, we describe our training procedure which we use for the performance comparison with the original wPINNs. Further notes and explanations detailing the implementation are located in Appendix B to keep the paper self-contained and ensure reproducibility of our numerical tests. Our python implementation is located at `https://git-ce.rwth-aachen.de/aidan.chaumet/wpinns`.

Our training procedure is described in Algorithm 1. It is essentially a standard generative adversarial neural network (GAN) training procedure including best model checkpoints, and as such many of the usual heuristics in the training of such network architectures apply. Note that lines 7 and 11 of the algorithm compute the loss function depending on the networks. These steps generate the computational graphs to backpropagate gradients using automatic differentiation for weight updates. We write these steps to represent a practical implementation in machine learning frameworks. Note that the list of hyperparameters given is not exhaustive.

5.2. Comparison Setup

Now, we describe our methodology for comparing the performance of the original and modified wPINNs. For the entirety of this section we consider $p = q = 2$ in the modified loss because this is most closely related to the original wPINNs.

As a measure of computational cost, we count the number of training epochs. The original wPINNs and the modified wPINNs require similar computational effort per epoch, because the most expensive part of each epoch is evaluating the PDE residual numerically which occurs equally often in original

Algorithm 1 Neural Network Training algorithm

```
Input: Initial data $u_0$, boundary data $g$, flux function $f$, strictly convex entropy-entropy flux pair
    $(\eta, q)$, Hyperparameters: $\tau_{\min}, \tau_{\max}, N_{\max}, \lambda, \gamma$
Output: Best networks $u_\theta^b, \varphi_\chi^b, \xi_\nu^b$
 1: Initialize the networks $u_\theta, \varphi_\chi, \xi_\nu$
 2: Initialize performance metric $\mathfrak{L}_{\text{avg}} \leftarrow \mathfrak{L}(u_\theta, \varphi_\chi, \xi_\nu)$
 3: Initialize best performance metric $\mathfrak{L}_{\text{best}} \leftarrow \infty$
 4: Generate collocation points $S_{\text{int}}, S_{\text{ic}}, S_{\text{bc}}$
 5: for $ep = 1, \dots, N_{\text{ep}}$ do
 6:     for $k = 1, \dots, N_{\max}$ do
 7:         Compute $\mathfrak{L}(u_\theta, \varphi_\chi, \xi_\nu)$
 8:         Update $\chi \leftarrow \chi + \tau_{\max} \nabla_\chi \mathfrak{L}(u_\theta, \varphi_\chi, \xi_\nu)$
 9:         Update $\nu \leftarrow \nu + \tau_{\max} \nabla_\nu \mathfrak{L}(u_\theta, \varphi_\chi, \xi_\nu)$
10:     end for
11:     Compute $\mathfrak{L}(u_\theta, \varphi_\chi, \xi_\nu)$
12:     Update $\theta \leftarrow \theta - \tau_{\min} \nabla_\theta \mathfrak{L}(u_\theta, \varphi_\chi, \xi_\nu)$
13:     Update performance indicator $\mathfrak{L}_{\text{avg}} \leftarrow (1-\gamma)\, \mathfrak{L}_{\text{avg}} + \gamma\, \mathfrak{L}(u_\theta, \varphi_\chi, \xi_\nu)$
14:     if $\mathfrak{L}_{\text{avg}} < \mathfrak{L}_{\text{best}}$ then
15:         $\mathfrak{L}_{\text{best}} \leftarrow \mathfrak{L}_{\text{avg}}$
16:         Save best networks $u_\theta^b, \varphi_\chi^b, \xi_\nu^b \leftarrow u_\theta, \varphi_\chi, \xi_\nu$
17:     end if
18: end for
```

and modified wPINNs. We do not compare wall-clock time, which may differ due to implementation details or choice of machine learning framework.

We make no comparison to classical PINNs because these cannot produce good approximate solutions. We also do not compare to conventional numerical methods because it is obvious that in one or two dimensions neural networks cannot compete, while in high dimensions conventional mesh-based methods are obviously unsuitable, so a like-to-like comparison is not possible.

We assess the accuracy of approximations u_θ using the relative space-time L^1 error given by

$$\mathcal{E}_{\text{r}}(u_\theta) = \frac{\int_{[0,\text{T})\times D} |u_\theta(t,x) - u(t,x)| \,\mathrm{d}x\,\mathrm{d}t}{\int_{[0,\text{T})\times D} |u(t,x)| \,\mathrm{d}x\,\mathrm{d}t}, \tag{5.1}$$

where u denotes the exact solution, in case it is easy to compute. The integrals are approximated using Monte-Carlo integration with 2^{17} points in the space-time domain. In cases where the exact solution is not readily available, we approximate it using a high-resolution finite volume scheme from PyClaw [21].

We perform an ensemble training procedure as in [7], however we do not investigate as large a range of hyperparameters. Instead, we try several hyperparameter configurations until we find parameters that give good results for the original wPINN algorithm and then use the same hyperparameters for training both approaches. Although there is no direct corresponding network for ξ_ν in the original wPINN approach, we match this network's hyperparameters to those of φ_χ. This keeps the two training procedures as comparable as possible.

As a learning rate schedule, we reduce the learning rate linearly by half over the total number of epochs. While we think that a more aggressive learning rate schedule would benefit the modified wPINNs, to be able to check our results against the original wPINN results from [7] we maintain their choice.

For both original and modified wPINNs, we track $\mathcal{E}_{\mathrm{r}}(u_\theta)$ for each network of the ensemble individually over the training epochs for comparison, sampled at multiple epochs during training.

Additionally, we also compare the quality of the *average network* predictions

$$u^*(t,x) = \frac{1}{N^*}\sum_{i=1}^{N^*} u_{\theta,i}(t,x), \tag{5.2}$$

for an ensemble $\{u_{\theta,i}\}_{i=1}^{N^*}$ consisting of N^* neural networks trained for the same hyperparameters, but different randomly chosen initial weights and biases. We compute $\mathcal{E}_{\mathrm{r}}(u^*)$ only at the final epoch of training.

Summarizing our numerical results, we list the average network prediction error in Table 5.1. In particular, the table shows that we reproduce the accuracy achieved by the original wPINNs in [7] and that the modified wPINNs match or outperform the original wPINNs in all examples, with significant gains especially for the less synthetic examples. In the following sections, we discuss each of the numerical experiments in more detail.

Table 5.1. Comparison of the relative error $\mathcal{E}_{\mathrm{r}}(u^*)$ of the average network prediction at the different times during training. The modified wPINNs are two times more accurate than the original wPINNs after less than 30% of the total epochs for the rarefaction wave and over 10 times more accurate for the sine initial data.

	original	modified
Standing shock (2000 Epochs)	0.105%	0.098%
Moving shock (2000 Epochs)	1.46%	1.36%
Rarefaction Wave (800 Epochs)	2.94%	1.57%
Rarefaction Wave (3000 Epochs)	1.45%	1.18%
Sine Wave (10000 Epochs)	21.4%	1.53%
Sine Wave (75000 Epochs)	5.03%	1.19%

5.3. **Standing Shock**

We begin by considering the Burgers equation on $[0, 0.5) \times [-1, 1]$ for the initial datum

$$u_0(x) = \begin{cases} 1 & \text{for } x \le 0, \\ -1 & \text{for } x > 0. \end{cases} \tag{5.3}$$

The corresponding exact solution is given by

$$u(t,x) = \begin{cases} 1 & \text{for } x \le 0, \\ -1 & \text{for } x > 0, \end{cases} \tag{5.4}$$

that is, a standing shock at $x = 0$.

We use the following hyperparameters which are the same for both procedures: $\tau_{\min} = 0.01, \tau_{\max} = 0.015$, $N_{\max} = 8$, $N_{\mathrm{ep}} = 2000$. The networks u_θ in both algorithms have $l = 6$ layers of width $w = 20$ with $\sigma(\,\cdot\,) = \tanh(\,\cdot\,)$ activation. The networks φ_χ and ξ_ν use 4 layers of width 10 with tanh activation function. We fix the convex entropy-entropy flux pair $\eta(u) = \frac{1}{2}u^2$ and $q(u) = \frac{1}{3}u^3$ for all following examples.

For the original wPINNs we adopt the penalty parameter $\lambda = 10$, as fixed during the numerical experiments in [7], while for the modified wPINNs $\lambda = 1$ seems to work better. Because we use different loss functions it is not sensible to keep this parameter matched. We set $\gamma = 0.3$ for the exponential averaging of past loss values. The original wPINN algorithm also requires a choice of reset frequency for the adversarial network which we choose as $r_f = 0.05$. Lastly, we compute $N^* = 16$ retrainings.

For both the original and modified wPINNs we use $N_{\text{int}} = 16384$ uniformly randomly sampled interior collocation points, $N_{\text{ic}} = 4096$ inital collocation points and $N_{\text{bc}} = 4096$ boundary collocation points. These points are randomly chosen anew for each retraining of neural networks and choice of algorithm, but stay fixed during the training of each single network.

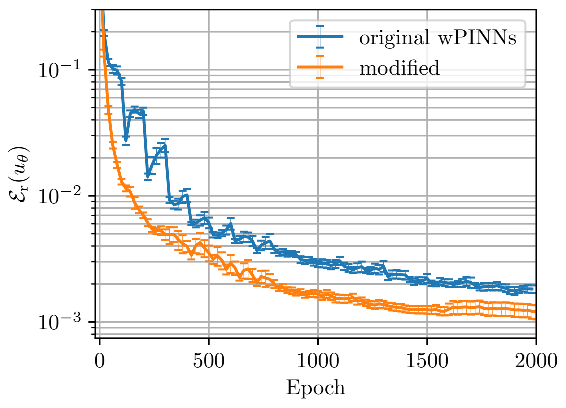
The average network prediction error is listed in Table 5.1. In this example we see that the performance of both methods is approximately the same. We give several reasons why this can be mainly attributed to the specific example below.

We plot the individual network's errors over the training epochs for the standing shock in Figure 5.1(a). The error bars show the standard error of the mean, that is $\frac{\sigma}{\sqrt{N^*}}$ with σ the standard deviation of the relative L^1 errors in the ensemble. There are a few small differences in the evolution of $\mathcal{E}_\text{r}$, such as the modified wPINNs giving a "smoother" evolution of the relative error over the epochs and having generally smaller statistical variations.

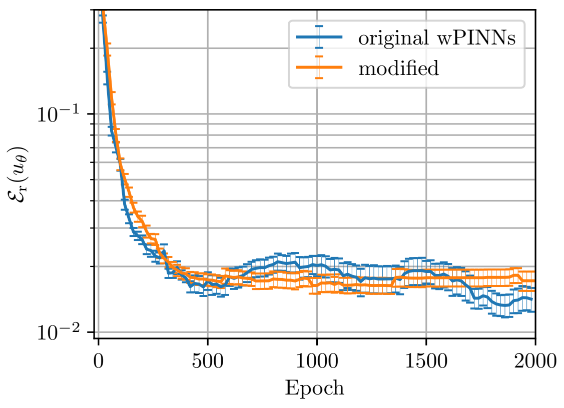
We attribute the smoother evolution of the relative error to the more stable norm estimation of our loss functional by avoiding the division by the $W^{1,2}$-seminorm of the approximating network and possibly the adjusted checkpointing procedure as well, as described in Appendix B.

Confirming our expectations, we see that both original and modified wPINNs perform well, showing that both methods are suitable for approximating shock solutions. The difference between the two methods is small in this example. The limiting factor for accuracy seems to be the network architecture. As depth and width are identical for either test, the accuracy will thus be similar. As the initial data is already a shock, the loss contributions from initial and boundary data heavily influence the overall learning of the solution profile. The additional PDE residual contribution only needs to ensure that the shock is placed at the right position, and because the shock is stationary this is straightforward, giving only a small contribution to the total loss.

The initial and boundary loss functions are identical for original and modified wPINNs. During training one can check that the majority of the overall loss for either approach is due to these contributions for this example. The different choice of penalty parameter λ only has a minor influence on the training procedure because the Adam algorithm for minimizing the loss is invariant with respect to scalar rescaling of the entire loss, which consists mainly of the initial- and boundary loss. As such,



(A) Standing shock



(B) Moving shock

Figure 5.1. Evolution of the mean value and standard error of $\mathcal{E}_\text{r}$ for the respective network ensembles.

the original and modified wPINNs are almost equivalent for this example and other situations where the shape of the solution does not differ much compared to the initial data.

5.4. **Moving Shock**

For completeness, we also consider a moving shock example. We choose

$$u_0(x) = \begin{cases} 1 & \text{for } x \leq 0, \\ 0 & \text{for } x > 0, \end{cases} \tag{5.5}$$

as an initial datum leading to the solution

$$u(t,x) = \begin{cases} 1 & \text{for } x \leq \frac{t}{2}, \\ 0 & \text{for } x > \frac{t}{2}, \end{cases} \tag{5.6}$$

which is a moving shock starting from $x = 0$, moving to the right with a speed of 0.5.

We choose the same training setup and hyperparameters as for the standing shock, including $N^* = 16$. As expected, the results for this example are structurally the same as in the standing shock example. We plot the results for the moving shock in Figure 5.1(b), and reference the average network prediction error in Table 5.1. The individual and averaged errors are larger for this example, but again the errors we get are in line with the expectations from the original wPINN experiments.

The rest of the discussion is analogous to the standing shock. However, because the shock is moving at a constant velocity, the PDE residual now essentially amounts to learning a single linear transformation to ensure correct placement of the solution profile learned from the initial data. This is slightly less straightforward than previously, resulting in a larger relative error and a small advantage for the modified wPINNs.

5.5. **Rarefaction Wave**

As another prototypical example, we consider a rarefaction wave on $[0, 0.5) \times [-1, 1]$ for the initial data

$$u_0(x) = \begin{cases} -1 & \text{for } x \leq 0, \\ 1 & \text{for } x > 0. \end{cases} \tag{5.7}$$

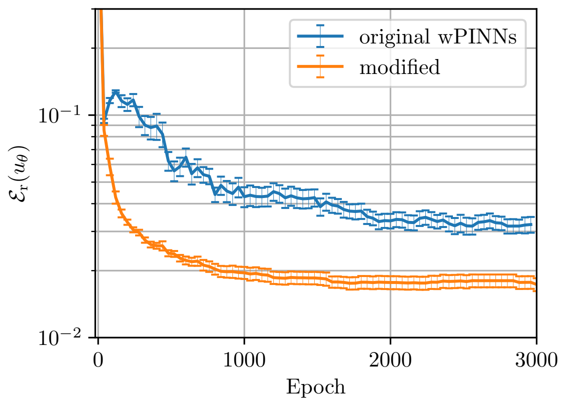
For this, the exact solution is given by a rarefaction wave

$$u(t,x) = \begin{cases} -1 & \text{for } x \leq -t, \\ \frac{x}{t} & \text{for } -t < x \leq t, \\ 1 & \text{for } x > t. \end{cases} \tag{5.8}$$

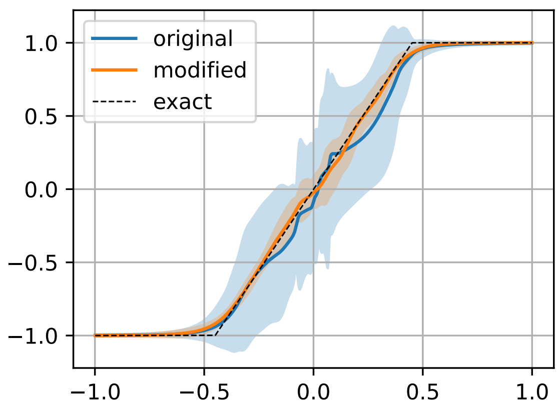
Without additional entropy conditions, weak solutions are non-unique for the given initial data. However, due to the entropy residual, the modified wPINNs find the correct entropy solution. We train the neural networks for this example using the same hyperparameters as in the previous examples, except that we train for a total of $N_{\text{ep}} = 3000$ epochs now.

The mean relative error of the ensemble as training progresses is shown in Figure 5.2(a). Because this example is no longer dominated by how quickly the network can learn the correct shape of the solution from the initial data, it is better-suited to illustrate the advantages of the modified wPINNs. In this example we see that the modified wPINNs provide a sizable advantage over the original wPINN algorithm.

For the errors of the original and modified wPINNs average network predictions, we refer to Table 5.1. After 800 epochs of training the modified wPINNs are already within 15% of their final accuracy. At this epoch, the modified wPINNs are roughly twice as accurate as the original wPINNs. The original wPINN network predictions have, on average, a $4.3\% \pm 0.42\%$ relative error while the



(A) Evolution of the mean value and standard error of $\mathcal{E}_{\mathrm{r}}$ for the respective network ensembles. After 800 epochs, the modified wPINNs are already within 15% of the final relative error, while the original wPINNs are still 35% worse than their final relative error.



(B) Comparison of average network predictions for original and modified wPINNs at $t = 0.45$ after $N_{\mathrm{ep}} = 800$ epochs. Shaded bands are 2σ-standard deviations of the average network prediction. We refer to the text for discussion.

Figure 5.2. Rarefaction wave with shock initial data

modified wPINNs have an error of only $2.0\% \pm 0.11\%$. Prolonged training is able to close the accuracy gap somewhat, as the modified wPINNs are closer to finishing learning and additional epochs enable the original wPINNs to catch up slightly. However, they are not able to completely close the accuracy gap to the modified wPINNs even after 3.5x longer training.

In Figure 5.2(b) we show the ensemble averaged prediction after 800 epochs of training for the original and modified wPINN algorithm at $t = 0.45$. We see that the accuracy of the original wPINNs is much lower, such that one can easily see the deviations from the reference solution. Further, the original wPINN algorithm produces spurious oscillations when approximating the solution to this problem. While both the modified loss and the original wPINN loss penalize spurious oscillations as argued in Section 3, we observe that the modified approach seems to penalize this more effectively, so we do not observe significant oscillations for its results.

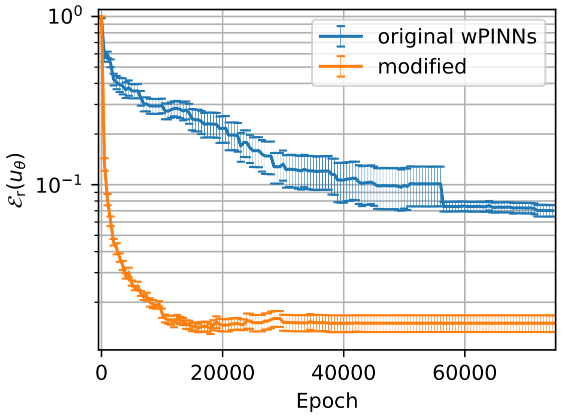
Finally, we observe that the standard deviation of the average network prediction is much smaller for the modified wPINNs, showing that the modified wPINNs are more stable during training. This is an advantage, because every network in our ensemble is an accurate approximation. The *worst-performing* network in the modified wPINNs ensemble at epoch 800 achieves a relative L^1 error of 2.91%, which is as good as the *averaged* network prediction of the original wPINNs at 2.94% as shown in Table 5.1. Thus, the smaller standard deviations for the modified wPINNs mean that one could also forego ensemble training (or use smaller ensembles), reducing computational costs.

5.6. **Sine Wave**

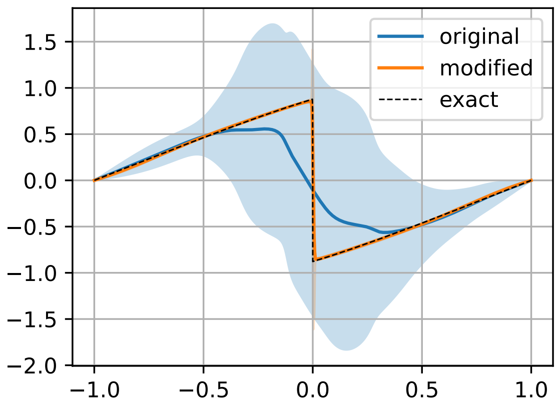
Lastly we consider the Burgers equation for sine initial data $u_0(x) = -\sin(\pi x)$ on $[0, 1) \times [-1, 1]$ and $u(t, -1) = u(t, 1) = 0$ for $t \in [0, 1)$. This example is significantly more challenging to solve to high precision using neural networks, because unlike previous examples it contains shock formation from smooth initial data. This means the network approximation has to steepen over time, developing into a shock separating two rarefactions. This example is also studied in [7] and represents the typical performance of both original and modified wPINNs better than the more synthetic tests of a single discontinuity or a pure rarefaction example.

Because this example is more challenging, and to reproduce the simulation setup in [7], instead of uniformly randomly sampled points on the domain we use low-discrepancy Sobol points. These approximate the numerical integrals more precisely than previous uniformly randomly sampled points, improving network performance.

In our preliminary experiments we find the hyperparameter choices from the previous two examples to perform best for the original wPINN algorithm again. However, the exact solution is more complicated, so we increase the number of epochs to $N_{\text{ep}} = 75000$ to match the training outlined in [7] for this example. However, we find that modified wPINNs require significantly less epochs to achieve good results during our numerical experiments. Because we train for more epochs, we reduce γ to 0.015 for the exponential loss averages.



(A) Evolution of the mean value and standard error of $\mathcal{E}_\text{r}$ for the respective network ensembles. The modified wPINNs are within 15% of their final accuracy after 10000 epochs, while the original wPINNs are still four times less accurate than they are at the end of training.



(B) Comparison of average network predictions for original and modified wPINNs at $t = 0.75$ after $N_{\text{ep}} = 10000$ epochs. Shaded bands are 2σ-standard deviations of the average network prediction. We refer to the text for discussion.

FIGURE 5.3. Solution and training progress for sine initial data

As before, we show the mean relative error of the ensemble over the training epochs in Figure 5.3(a). In this example we see a very noticeable difference between the original and the modified wPINN algorithms. The mean value of $\mathcal{E}_\text{r}$ at the final training epoch for the original wPINNs is $7.0\% \pm 0.6\%$, while the modified algorithm gives a mean relative error of only $1.5\% \pm 0.17\%$. We also compare results after 10000 epochs, when the modified wPINNs are within 15% of their final accuracy. The modified wPINNs achieve an accuracy of $1.73\% \pm 0.11\%$, while the original wPINNs have an accuracy of only $29.5\% \pm 2.9\%$ on average. The drop in the mean relative error for the original wPINNs after about 55000 epochs alongside a large reduction in the standard error is due to a single poorly-performing outlier network improving to match the performance of the remaining ensemble at this point in training. This is very visible given the limited ensemble size of $N^* = 16$ retrainings.

The average network predictions after 10000 epochs are compared in Figure 5.3(b). At this time during training, the original wPINNs have a relative error of $\mathcal{E}_\text{r}(u^*_{\text{orig}}) = 21.4\%$, while the modified wPINNs already achieve an accuracy of $\mathcal{E}_\text{r}(u^*_{\text{mod}}) = 1.53\%$. At the end of training, the original wPINNs have a relative error of $\mathcal{E}_\text{r}(u^*_{\text{orig}}) = 5.03\%$, which is larger but comparable to the results from [7]. For the modified wPINNs we find $\mathcal{E}_\text{r}(u^*_{\text{mod}}) = 1.19$ % instead, which is better than the results from [7]. The modified wPINNs have finished learning completely after about 20000 epochs, as their relative error plateaus. Any prolonged training benefits only the original wPINNs. Yet, even after the full

75000 epochs, the original wPINNs are not as accurate as the modified wPINNs. Again, the modified wPINN ensemble has a much smaller standard deviation than the original wPINN ensemble. This behavior persists even after the full training period of 75000 epochs and is even more pronounced than in the previous example.

Note that for the modified wPINNs algorithm, its final level of accuracy is roughly in line with the previous example, with the main difference being that it takes more epochs to reach this level of accuracy. In our tests we saw that increasing the number of collocation points and choosing a different learning rate schedule can further improve results, while the networks themselves appear to be able to approximate the solution of the underlying problem to higher accuracy without changing the network size, indicating that these hyperparameters restrict the final precision most here.

6. Numerical Results - Extensions

6.1. Weak Boundary Conditions

We test our extension for weak Dirichlet boundary data. We solve the Burgers equation on $[0, 0.8) \times [-0.5, 0.5]$ for the initial data

$$u_0 = \begin{cases} 2 & \text{for } x \leq 0, \\ 0 & \text{for } x > 0, \end{cases} \tag{6.1}$$

and the boundary data $g(t, -0.5) = 2 - 2t$ and $g(t, 0.5) = -1$ for all $t \in [0, 0.8)$. In this case, the left-side boundary is part of the inflow boundary the entire time, while the right-side boundary initially starts as inflow boundary and then switches to outflow boundary as the shock propagates out of the domain.

For the training of this problem, as described in Section 4.3, we subdivide the time domain into six equally large time slices because long time domains can be problematic when using weak boundary conditions. Then we begin by training a neural network on the first time slice $t \in [0, 2/15]$ and use the evaluation of the first neural network at $t = 2/15$ as the initial data for a separate neural network trained on $t \in [2/15, 4/15]$ and repeat this process until the final time is reached.

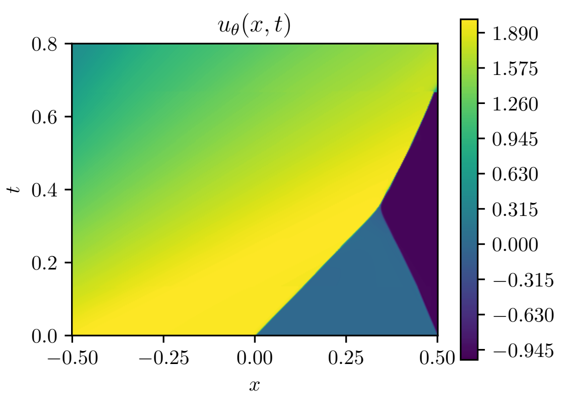
For training we use the following hyperparameters: $\tau_{\min} = 0.005$, $\tau_{\max} = 0.009$, $N_{\max} = 8$, $N_{\text{ep}} = 5000$. The network u_θ has $l = 6$ layers with a width of $w = 20$ and tanh-activation, while the networks φ_χ and ξ_ν use 4 layers of width 14 and tanh-activation. We use $N_{\text{int}} = 32768$ points per time slice and $N_{\text{ic}} = N_{\text{bc}} = 4096$ points for initial- and boundary conditions on each subdomain. We do not perform any ensemble training. The hyperparameters were not exhaustively tuned to achieve best performance.

The neural network approximation after training is displayed in Figure 6.1(a) and the pointwise error to the exact solution in Figure 6.1(b).

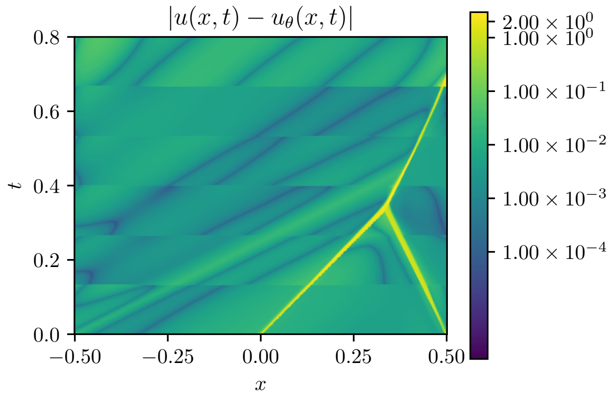
We see that both imposing the boundary condition when valid, as well as *not* enforcing the boundary data on the right as the shock exits the domain, work correctly. In particular, the shock exits the domain around $t = 0.7$. Note that due to understanding the boundary conditions in a weak sense this does not require any prior knowledge but rather results from simulation. The logarithmically scaled pointwise error shows the discontinuities at the interface between two time slices clearly and that coupling the subdomains does not incur large errors. Our approach solves the problem with good accuracy, leading to a final relative L^1-error of $\mathcal{E}_{\text{r}}(u_\theta) = 1.21\%$ across the entire domain.

6.2. Compressible Euler Equations

In this section we apply the modified wPINNs to the compressible Euler equations for the Sod shock tube. Since the Kruzhkov entropies are not suitable for systems, the original wPINNs cannot be applied to this example, so we cannot make a comparison. The compressible Euler equations in primitive



(A) Neural network solution. The network automatically decide when the right boundary is part of the inflow- or outflow boundary.



(B) Pointwise error of the neural network approximation. The logarithmic scale of the colorbar makes the small discontinuities in the pointwise error at the interfaces between the time slices visible.

Figure 6.1. Burgers equation with weak boundary conditions.

variables are given by

$$\begin{cases} \rho_t + (\rho v)_x = 0, & (6.2)\\ (\rho v)_t + (\rho v^2 + p)_x = 0, & (6.3)\\ E_t + (v(E+p))_x = 0, & (6.4) \end{cases}$$

in $[0, \mathrm{T}) \times \mathbb{R}$ where ρ is the density, v is the velocity and p is the pressure. For Sod's problem, the pressure relates to the internal energy e of the gas as $p = \rho e(\gamma - 1)$ with γ the adiabatic exponent of the gas. We consider $\gamma = 1.4$, as it would be for an ideal diatomic gas. Lastly, the total energy E is given by $E = \rho e + \frac{1}{2}\rho v^2$. As in [43], our network parameterizes the primitive variables (ρ, p, v) instead of conservative variables $(\rho, \rho v, E)$ because this avoids divisions by zero when computing $v = (\rho v)/\rho$. However, the system is still given in conservative form, to align with our previous arguments in favor of using weak norms.

Sod's problem then consists of solving the Euler equations for the Riemann problem with initial data

$$\begin{cases} (\rho, p, v) = (1.0, 1.0, 0.0) & \text{for } t = 0,\ \ x < 0.5, \ \text{ and} \quad (6.5)\\ (\rho, p, v) = (0.125, 0.1, 0.0) & \text{for } t = 0,\ \ x \geq 0.5. \quad (6.6) \end{cases}$$

An exact solution is available for comparison with the numerical results [42].

We parameterize our approximations for ρ, v and p each using a separate neural network. While one could also use a single neural network to describe the entire state vector (ρ, v, p), this holds the disadvantage that discontinuities in one state variable tend to lead to poor performance in the other state variables, especially for smaller networks, because all components share the same overall network. This makes learning significantly more difficult. Likewise, we have three separate networks responsible for the weak norm estimation, one per equation of the Euler system.

For training, we use a PDE residual loss akin to equation (4.6) employing integration by parts on the x-derivatives for each of (6.2)–(6.4), and sum up the contributions per equation. We use $q = 2$ in the modified loss and this seems to work well. We give the complete loss function for this example in Appendix C.

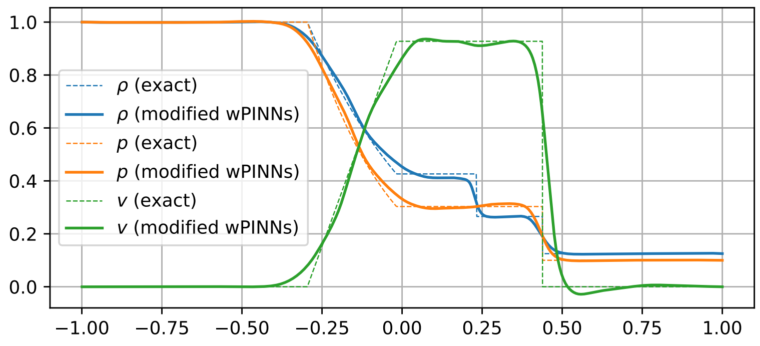


FIGURE 6.2. Density, pressure and velocity of the neural network approximation for Sods shock tube at $t = 0.25$. We see that all the different regions of the exact solution are resolved and there is only mild smearing around the different discontinuities.

Lastly, for the entropy residual part in our approach we use the entropy-entropy flux pair

$$(\eta, q) := (\rho S, \rho v S), \tag{6.7}$$

with $S := \ln(\rho^\gamma/p)$ and enforce the usual entropy inequality [44]. Initial- and boundary conditions are incorporated into the loss through their standard L^2 contributions, completely analogously to the scalar case. We do not perform any ensemble training for this example. Our neural networks for the approximation of ρ, v and p consist of 6 fully connected layers of 45 neurons each. This is larger than in our previous scalar examples, but we find that this size is necessary to adequately capture the two discontinuities that will arise in the density profile. As activation function we use a scaled softplus function given by $\text{softplus}(40; x) := \frac{1}{40}\log(1 + \exp(40 \cdot x))$, which serves as a smooth approximation to the ReLu-function. The scaling factor 40 was chosen arbitrarily. We find that this smooth approximation performs better than the ReLu activation function, presumably because having continuous derivatives makes learning easier. We apply the unscaled softplus-function to the output of our density- and pressure networks. The entropy-entropy flux pair is not well-defined for negative density or pressure, however basic random initialization of the network weights cannot ensure that this does not occur. This step then guarantees that the output of these networks is always positive and is important for the early steps during training. Later into the training, the approximation is reasonably close to the true solution, so it has non-negative pressure and density anyway.

All other networks in this example have 6 layers with 24 neurons each and also use $\text{softplus}(40; x)$ as layer activation function.

We solve the shock tube on $[0, 0.4] \times [-1, 1]$. We use $N_{\text{int}} = 65536$ Sobol points in the interior of the domain, and $N_{\text{ic}} = N_{\text{bc}} = 8096$ points for initial and boundary condition respectively. We train for a total of $N_{\text{ep}} = 42000$ epochs using a learning rate of $\tau_{\min} = 0.005$ and $\tau_{\max} = 0.014$. We choose a learning rate schedule that linearly reduces the learning rate by a factor of 30 over the course of training, because we need to minimize the loss to fairly high precision. We optimize using the Adam optimizer, and set the hyperparameters $\beta_1 = 0.2$ and $\beta_2 = 0.99$. Using lower values of β_1 and β_2 than "default" is common for training adversarial networks [15].

We show the final trained network at time $t = 0.25$ in Figure 6.2, which is representative of the performance over the entire space-time domain. We compute the relative L^1 in space-time error for the density, pressure and velocity respectively and find $\mathcal{E}_\text{r}(\rho_{\text{mod}}) = 1.38\%$, $\mathcal{E}_\text{r}(p_{\text{mod}}) = 1.47\%$ and $\mathcal{E}_\text{r}(v_{\text{mod}}) = 3.77\%$.

7. Discussion

Approximating nonlinear hyperbolic conservation laws using conventional PINNs fails when the exact solution is discontinuous and PDE residuals are minimized in the L^2 norm. The failure of conventional PINNs is structural and cannot be remedied by using larger networks or training more, because the residual is measured in an unsuitable norm. This class of problems requires using *weak norms* instead. The wPINN strategy from [7] enables the accurate approximation of entropy solutions to scalar conservation laws by computing weak norms. Test functions for the PDE residual are parameterized with neural networks, leading to an adversarial min-max problem structure which may be solved with standard deep learning techniques.

However, the training of wPINNs is computationally more expensive than that of conventional PINNs, because of the min-max problem structure. Using the supremum-based definition of the weak norms leads to a challenging maximization problem.

We improve on this by providing a new strategy for approximating weak norms. The maximizer can be found as the solution to a dual elliptic problem. This avoids normalizing the test function. While this still leads to a min-max formulation of the loss, it has a simpler structure that makes training wPINNs faster.

Beyond the changes to computing dual norms, we modify several core parts of the original wPINN algorithm to accelerate training and enable extensions to systems of hyperbolic conservation laws. In the original wPINN algorithm, the Kruzhkov entropy residual has two important roles: enforcing the PDE in a weak sense and selecting the entropy solution. We split these roles into separate terms by introducing a PDE loss and an entropy loss with respect to a strictly convex entropy-entropy flux pair. This gives more flexibility to both terms. Crucially, this means we do not have to use the Kruzhkov family of entropies and can opt for a smoother entropy, which makes training easier. An additional benefit to not using the Kruzhkov entropies is that our modified wPINNs naturally extend to systems of hyperbolic conservation laws endowed with a convex entropy.

Numerical experiments illustrate the benefits of the modified wPINNs. Especially for more challenging problems we see large improvements expressed by lower mean relative errors after the same number of training epochs and better approximations at the end of training. The modified wPINNs finish training in significantly less epochs than the original wPINNs.

Finally, we show that the modified wPINNs are suitable for weak boundary conditions and systems of hyperbolic conservation laws. For scalar conservation laws we cover weak boundary conditions by enforcing an entropy-based inequality on the boundary of the domain. We verify that this approach is effective by solving the Burgers equation with weak boundary conditions. Our experiments show that this approach enables shocks to exit the computational domain without requiring any prior knowledge of when this occurs. Further, we give an example of modified wPINNs naturally being applicable to systems of hyperbolic conservation laws, by summing over the PDE residuals of each separate equation. We solve Sod's shock tube problem for the Euler equations successfully.

Appendix A. Time-dependent approximations with fixed zero

We extend the computations from Section 3 to another, somewhat more technical but also more general example. In this example we consider another class of time-dependent approximate solutions $\tilde{u}$ with

$$\begin{cases} \tilde{u}(t,x) \in [1-\epsilon, 1+\epsilon] & \text{for } x < -\epsilon, \\ \tilde{u}(t,x) \in [-1-\epsilon, -1+\epsilon] & \text{for } x \geq \epsilon, \\ \tilde{u}(t,x) \in [1+\epsilon, -1-\epsilon] & \text{for } -\epsilon \leq x \leq \epsilon \end{cases} \tag{A.1}$$

such that there exists an $\bar{x}$, independent of t, with $\tilde{u}(t,\bar{x}) = 0$. The existence of an $\bar{x}(t)$ follows from continuity, however the restriction to time-independent $\bar{x}$ is technical in nature. Further, we assume that $(\bar{x} - x)\tilde{u}(x) \geq 0$ for all $x \in D$. Splitting the integration over x at $\bar{x}$, we compute the L^1-Norm of the residual:

$$\begin{aligned}
\|\mathcal{R}[\tilde{u}]\|_{L^1([0,\mathrm{T})\times(-\epsilon,\epsilon))} &= \int_0^T \int_{-\epsilon}^{\epsilon} |f(\tilde{u})_x + \tilde{u}_t| \,\mathrm{d}x\,\mathrm{d}t \\
&\geq \int_0^T \left| \int_{-\epsilon}^{\bar{x}} f(\tilde{u})_x + \tilde{u}_t \,\mathrm{d}x \right| \mathrm{d}t + \left| \int_{\bar{x}}^{\epsilon} f(\tilde{u})_x + \tilde{u}_t \,\mathrm{d}x \right| \mathrm{d}t \\
&= \int_0^T \left| f(\tilde{u}(t,-\epsilon)) - \int_{-\epsilon}^{\bar{x}} \tilde{u}_t \,\mathrm{d}x \right| + \left| f(\tilde{u}(t,\epsilon)) + \int_{\bar{x}}^{\epsilon} \tilde{u}_t \,\mathrm{d}x \right| \mathrm{d}t \\
&\geq \int_0^T \Big(f(\tilde{u}(t,-\epsilon)) + \int_{-\epsilon}^{\bar{x}} (\tilde{u}_t)^{\ominus} \,\mathrm{d}x - \int_{-\epsilon}^{\bar{x}} (\tilde{u}_t)^{\oplus} \,\mathrm{d}x \\
&\qquad + f(\tilde{u}(t,\epsilon)) - \int_{\bar{x}}^{\epsilon} (\tilde{u}_t)^{\ominus} \,\mathrm{d}x + \int_{\bar{x}}^{\epsilon} (\tilde{u}_t)^{\oplus} \,\mathrm{d}x \Big) \,\mathrm{d}t\,.
\end{aligned} \tag{A.2}$$

where $(\,\cdot\,)^{\ominus}$ and $(\,\cdot\,)^{\oplus}$ denote the negative and positive part of a function respectively. Then, because $\bar{x}$ is independent of t, we may swap the order of integration and use the fact that

$$\left| \int_0^T (\tilde{u}_t)^{\oplus}(t,x) - (\tilde{u}_t)^{\ominus}(t,x) \,\mathrm{d}t \right| = \left| \int_0^T \tilde{u}_t(t,x) \,\mathrm{d}t \right| = |\tilde{u}(T,x) - \tilde{u}(0,x)| \leq 1 + \epsilon. \tag{A.3}$$

Inserting this into the last line of (A.2) gives

$$\begin{aligned}
\|\mathcal{R}[\tilde{u}]\|_{L^1([0,\mathrm{T})\times(-\epsilon,\epsilon))} &\geq T(1-\epsilon)^2 - \int_{-\epsilon}^{\bar{x}} 1 + \epsilon \,\mathrm{d}x - \int_{\bar{x}}^{\epsilon} 1 + \epsilon \,\mathrm{d}x \\
&\geq T(1-\epsilon)^2 - 2\epsilon(1+\epsilon),
\end{aligned} \tag{A.4}$$

and we may conclude our estimate analogously to equation (3.12), giving a lower bound for the L^2-in-space-time norm of the residual scaling as $\frac{1}{\sqrt{\epsilon}}$.

Appendix B. Further Implementation Notes

To improve transparency and reproducibility of our results, we outline some common and useful heuristics training generative adversarial networks that we employed in our numerical experiments.

B.1. Gradient Descent Algorithms

In the general Algorithm 1, we describe regular gradient descent weight updates. Instead, any other gradient-based optimization algorithm may be used. We have experienced good results using the Adam algorithm [23] and its AMSGrad variant [38], which are both extremely popular choices for training neural networks due to their oftentimes faster minimization of the training loss.

B.2. Loss Computation

While we think of $\mathcal{L}(u_\theta, \varphi_\chi, \xi_\nu)$ as the loss function used for minimization with respect to u_θ and maximization with respect to φ_χ and ξ_ν, one does not need to compute the entire loss for each of the optimization steps. As discussed during the definition of our loss (4.10), it possesses an additive decomposition into several parts depending only on one or two networks. We do not evaluate parts of the loss when they do not contribute to the gradients for weight updates, avoiding unnecessary backpropagations.

B.3. **Learning Rates and Scheduling**

The learning rates $\tau_{\min}$ and $\tau_{\max}$ play an important role in training and choosing them adequately has a big impact on both training time and resulting network parameters. On the one hand, larger learning rates are desirable because larger gradient descent steps require less overall steps, but on the other hand, because the loss function is typically non-convex, using a large and fixed learning rate often results in worse final network configurations even when the update is stable.

Instead of fixed learning rates, it is common to employ *learning rate scheduling*. It is typical to start with a large initial learning rate and then reduce it over the course of training, commonly by one or more orders of magnitude as training progresses. Further, one may *cycle* learning rates in a schedule [41], that is, intermittently increase the learning rate again to attempt to escape local minima or speed up training when the network is in a "plateau" of the loss landscape, which is slow to traverse with small learning rates.

In our experience, cyclical learning rate scheduling was very effective at obtaining more precise results. However, our numerical experiments only use a basic linear learning rate schedule to match the original wPINNs [7], when comparing performance.

Note that it is generally unclear whether both the estimation networks φ_χ and ξ_ν should use the same learning rate $\tau_{\max}$. If learning proves difficult for any of these networks one should consider using separate learning rates and number of maximization steps for either.

B.4. **Network Checkpointing**

An important technique during the training of neural networks is *checkpointing*, i.e. saving well-performing neural network weights and returning these instead of the final weights obtained after all training epochs are completed.

As a heuristic for determining network performance, one option is the neural network loss. We do not choose the loss directly but rather track an exponential average $\mathfrak{L}_{\text{avg}}$ decaying at a rate of $1-\gamma$, where $\gamma \in (0,1)$ is some parameter one is free to choose. Updating the performance indicator $\mathfrak{L}_{\text{avg}}$ is done using the loss from before the weight update to u_θ, however typical update steps only change this quantity slightly per epoch and doing so avoids one additional loss computation per epoch, making this convenient in practice.

The exponential averaging is non-standard, however we believe it is more sensible in adversarial settings than just tracking the epoch loss directly. Indeed, the solution network u_θ may update its weights in a fashion that confuses the adversarial networks, such that they grossly underestimate the correct value of the loss temporarily. This leads to several epochs where the loss is low and the solution network then performs a few bad weight updates before the adversarial networks recover and give accurate estimations again. Afterwards, one correctly observes an increased loss due to the poor weight updates of previous iterations.

Thus, it is disadvantageous to save single epochs where the loss is lowest, because this favors saving networks where the adversarial networks are not working properly. Tracking an exponential average of the loss mitigates this issue, leaving only the desired property of the averaged loss decreasing as the approximation to the solution improves, when the adversarial networks are working well.

Appendix C. **Complete loss for the Euler equations**

For the Euler system, we have three separate networks approximating ρ, p and v. Further, we have three test functions φ_1, φ_2 and φ_3, one for each equation of the Euler system. Finally we have a single neural network ξ to test the entropy equation.

The overall loss for the Euler system is obtained by summing up the modified wPINNs-style losses for each equation of the system. For brevity we define $E(\rho, v, p) := \frac{1}{2}\rho v^2 + \frac{p}{(\gamma-1)}$ and $S(\rho, p) := \ln\left(\frac{\rho^\gamma}{p}\right)$. On the interior of the domain, we obtain the following loss function:

$$\begin{aligned}
&\mathcal{L}_{\text{int}}(\rho, p, v, \varphi_1, \varphi_2, \varphi_3, \xi) \\
&\quad = \int_{[0,T)}\int_D \rho_t\varphi_1 - (\rho v)\,(\varphi_1)_x \,\mathrm{d}x\,\mathrm{d}t - \frac{1}{2}\int_{[0,T)}\int_D (\varphi_1)_x^2 \,\mathrm{d}x\,\mathrm{d}t \\
&\qquad + \int_{[0,T)}\int_D (\rho v)_t\varphi_2 - \left(\rho v^2 + p\right)(\varphi_2)_x \,\mathrm{d}x\,\mathrm{d}t - \frac{1}{2}\int_{[0,T)}\int_D (\varphi_2)_x^2 \,\mathrm{d}x\,\mathrm{d}t \\
&\qquad + \int_{[0,T)}\int_D E(\rho, v, p)_t\varphi_3 - (v(E(\rho, v, p) + p))\,(\varphi_3)_x \,\mathrm{d}x\,\mathrm{d}t - \frac{1}{2}\int_{[0,T)}\int_D (\varphi_3)_x^2 \,\mathrm{d}x\,\mathrm{d}t \\
&\qquad + \int_{[0,T)}\int_D ((\rho S(\rho, p))_t\xi + (\rho v S(\rho, p))\xi_x)^{\oplus} \,\mathrm{d}x\,\mathrm{d}t - \frac{1}{2}\int_{[0,T)}\int_D (\xi)_x^2 \,\mathrm{d}x\,\mathrm{d}t
\end{aligned} \tag{C.1}$$

Additional contributions for the boundary- and initial conditions are included in the standard L^2-sense, giving a total loss of

$$\mathcal{L}(\rho, p, v, \varphi_1, \varphi_2, \varphi_3, \xi) = \mathcal{L}_{\text{int}}(\rho, p, v, \varphi_1, \varphi_2, \varphi_3, \xi) + \lambda\left(\mathcal{L}_{\text{bc}}(\rho, p, v) + \mathcal{L}_{\text{ic}}(\rho, p, v)\right). \tag{C.2}$$

This is discretized again using Monte-Carlo sampling points. The overall method is then given by

$$(\hat{\rho}, \hat{p}, \hat{v}, \hat{\varphi}_1, \hat{\varphi}_2, \hat{\varphi}_3, \hat{\xi}) = \operatorname{argmin}_{\rho,p,v} \operatorname{argmax}_{\varphi_1,\varphi_2,\varphi_3,\xi} \mathcal{L}(\rho, p, v, \varphi_1, \varphi_2, \varphi_3, \xi). \tag{C.3}$$

References

[1] Claude Bardos, Alain-Yves Leroux, and Jean-Claude Nedelec. First order quasilinear equations with boundary conditions. *Commun. Partial Differ. Equations*, 4(9):1017–1034, 1979.

[2] Pavel B. Bochev and Max D. Gunzburger. *Least-Squares Finite Element Methods.* Applied Mathematical Sciences. Springer, 2009.

[3] Constantine M. Dafermos. *Hyperbolic conservation laws in continuum physics*, volume 325 of *Grundlehren der Mathematischen Wissenschaften.* Springer, fourth edition, 2016.

[4] Camillo De Lellis, Felix Otto, and Michael Westdickenberg. Minimal entropy conditions for Burgers equation. *Q. Appl. Math.*, 62(4):687–700, 2004.

[5] Tim De Ryck, Samuel Lanthaler, and Siddhartha Mishra. On the approximation of functions by tanh neural networks. *Neural Netw.*, 143:732–750, 2021.

[6] Tim De Ryck and Siddhartha Mishra. Generic bounds on the approximation error for physics-informed (and) operator learning. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pages 10945–10958. Curran Associates, Inc., 2022.

[7] Tim De Ryck, Siddhartha Mishra, and Roberto Molinaro. wPINNs: Weak Physics Informed Neural Networks for Approximating Entropy Solutions of Hyperbolic Conservation Laws. *SIAM J. Numer. Anal.*, 62(2):811–841, 2024.

[8] Waleed Diab and Mohammed Al Kobaisi. PINNs for the Solution of the Hyperbolic Buckley-Leverett Problem with a Non-convex Flux Function. `https://arxiv.org/abs/2112.14826`, 2021.

[9] George Dinca, Petru Jebelean, and Jean L. Mawhin. Variational and topological methods for Dirichlet problems with p-Laplacian. *Port. Math. (N.S.)*, 58(3):339–378, 2001.

[10] Gamini Dissanayake and Nhan Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Commun. Numer. Methods Eng.*, 10(3):195–201, 1994.

[11] Weinan E and Bing Yu. The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. *Commun. Math. Stat.*, 6(1):1–12, 2018.

[12] Antonio Ferrer-Sánchez, José D. Martín-Guerrero, Roberto Ruiz de Austri-Bazan, Alejandro Torres-Forné, and José A. Font. Gradient-annihilated PINNs for solving Riemann problems: Application to relativistic hydrodynamics. *Comput. Methods Appl. Mech. Eng.*, 424: article no. 116906, 2024.

[13] Ulrik S. Fjordholm, Samuel Lanthaler, and Siddhartha Mishra. Statistical Solutions of Hyperbolic Conservation Laws: Foundations. *Arch. Ration. Mech. Anal.*, 226(2):809–849, 2017.

[14] Cedric G. Fraces and Hamdi Tchelepi. Physics Informed Deep Learning for Flow and Transport in Porous Media. In *SPE Reservoir Simulation Conference*, 2021.

[15] Ian Gemp and Brian McWilliams. The unreasonable effectiveness of adam on cycles. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada*, 2019.

[16] Jean-Luc Guermond. A Finite Element Technique for Solving First-Order PDEs in L^p. *SIAM J. Numer. Anal.*, 42(2):714–737, 2004.

[17] Jean-Luc Guermond, Fabien Marpeau, and Bojan Popov. A fast algorithm for solving first-order PDEs by L^1-minimization. *Commun. Math. Sci.*, 6(1):199–216, 2008.

[18] Jean-Luc Guermond and Bojan Popov. L^1-minimization methods for Hamilton–Jacobi equations: the one-dimensional case. *Numer. Math.*, 109(2):269–284, 2008.

[19] Zheyuan Hu, Ameya D. Jagtap, George Em Karniadakis, and Kenji Kawaguchi. When Do Extended Physics-Informed Neural Networks (XPINNs) Improve Generalization? *SIAM J. Sci. Comput.*, 44(5):A3158–A3182, 2022.

[20] Chen Jingrun. A Comparison Study of Deep Galerkin Method and Deep Ritz Method for Elliptic Problems with Different Boundary Conditions. *Commun. Math. Res.*, 36(3):354–376, 2020.

[21] David I. Ketcheson, Kyle T. Mandli, Aron J. Ahmadia, Amal Alghamdi, Manuel Quezada de Luna, Matteo Parsani, Matthew G. Knepley, and Matthew Emmett. PyClaw: Accessible, Extensible, Scalable Tools for Wave Propagation Problems. *SIAM J. Sci. Comput.*, 34(4):C210–C231, 2012.

[22] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. Variational Physics-Informed Neural Networks For Solving Partial Differential Equations. `https://arxiv.org/abs/1912.00873`, 2019.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[24] Cezar I. Kondo and Philippe G. LeFloch. Measure-valued solutions and well-posedness of multi-dimensional conservation laws in a bounded domain. *Port. Math. (N.S.)*, 58(2):171–193, 2001.

[25] Isaac E. Lagaris, Aristidis Likas, and Dimitrios I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.*, 9(5):987–1000, 1998.

[26] Isaac E. Lagaris, Aristidis Likas, and D. G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Trans. Neural Netw.*, 11(5):1041–1049, 2000.

[27] John E. Lavery. Nonoscillatory solution of the steady-state inviscid burgers' equation by mathematical programming. *J. Comput. Phys.*, 79(2):436–448, 1988.

[28] Yulei Liao and Pingbing Ming. Deep Nitsche Method: Deep Ritz Method with Essential Boundary Conditions. *Commun. Comput. Phys.*, 29(5):1365–1384, 2021.

[29] Li Liu, Shengping Liu, Hui Xie, Fansheng Xiong, Tengchao Yu, Mengjuan Xiao, Lufeng Liu, and Heng Yong. Discontinuity Computing with Physics-Informed Neural Network. *J. Sci. Comput.*, 98(1): article no. 22, 2024.

[30] Emmanuel Lorin and Arian Novruzi. Non-diffusive neural network method for hyperbolic conservation laws. *J. Comput. Phys.*, 513: article no. 113161, 2024.

[31] Kjetil O. Lye, Siddhartha Mishra, and Deep Ray. Deep learning observables in computational fluid dynamics. *J. Comput. Phys.*, 410: article no. 109339, 2020.

[32] Piotr Minakowski and Thomas Richter. A priori and a posteriori error estimates for the Deep Ritz method applied to the Laplace and Stokes problem. *J. Comput. Appl. Math.*, 421: article no. 114845, 2023.

[33] Siddhartha Mishra and Roberto Molinaro. Estimates on the generalization error of physics-informed neural networks for approximating PDEs. *IMA J. Numer. Anal.*, 43(1):1–43, 2022.

[34] Guofei Pang, Lu Lu, and George Em Karniadakis. fPINNs: Fractional Physics-Informed Neural Networks. *SIAM J. Sci. Comput.*, 41(4):A2603–A2626, 2019.

[35] Evgueni Yu. Panov. Uniqueness of the solution of the Cauchy problem for a first order quasilinear equation with one admissible strictly convex entropy. *Math. Notes*, 55(5):517–525, 1994.

[36] Ravi G. Patel, Indu Manickam, Nathaniel A. Trask, Mitchell A. Wood, Myoungkyu Lee, Ignacio Tomas, and Eric C. Cyr. Thermodynamically consistent physics-informed neural networks for hyperbolic systems. *J. Comput. Phys.*, 449: article no. 110754, 2022.

[37] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. 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.

[38] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the Convergence of Adam and Beyond. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[39] Yeonjong Shin, Zhongqiang Zhang, and George Em Karniadakis. Error Estimates of Residual Minimization using Neural Networks for Linear PDEs. *J. Mach. Learn. Model. Comput.*, 4(4):73–101, 2023.

[40] Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018.

[41] Leslie N. Smith. Cyclical Learning Rates for Training Neural Networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

[42] Gary A. Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *J. Comput. Phys.*, 27(1):1–31, 1978.

[43] Erik Laurin Strelow, Alf Gerisch, Jens Lang, and Marc E. Pfetsch. Physics informed neural networks: A case study for gas transport problems. *J. Comput. Phys.*, 481: article no. 112041, 2023.

[44] Magnus Svärd. Entropy solutions of the compressible Euler equations. *BIT Numer. Math.*, 56(4):1479–1496, 2016.

[45] Chuwei Wang, Shanda Li, Di He, and Liwei Wang. Is L^2 Physics Informed Loss Always Suitable for Training Physics Informed Neural Network? *Adv. Neural Inf. Process. Syst.*, 35:8278–8290, 2022.

[46] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality for training physics-informed neural networks. *Comput. Methods Appl. Mech. Eng.*, 421: article no. 116813, 2024.

[47] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks. *SIAM J. Sci. Comput.*, 43(5):A3055–A3081, 2021.

[48] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective. *J. Comput. Phys.*, 449: article no. 110768, 2022.