ANN-Flux Method to Nonlinear Conservation Laws Flux with Two Inflection Points

Marcelo M. Mello[*], **Arthur M. Espírito Santo**, **Pedro H. A. Konzen**,
Instituto de Matemática e Estatística, IME, UFRGS,
Porto Alegre, RS
E-mail: marcelomokwademello@gmail.com, arthurmes@ufrgs.br, pedro.konzen@ufrgs.br.

Abstract: Artificial neural networks (ANNs) have been successfully applied to solve partial differential equations (PDEs). Conservation laws are PDEs that arise in a variety of fields such as fluid dynamics, traffic modeling, and fluid flow through porous media. We propose the new ANN-Flux method to solve the Riemann problem for nonlinear scalar conservation laws. The ANN-Flux method is designed for complex fluxes, which are costly to evaluate, differentiate, or invert. Based on the entropic solution form, the method applies two ANNs to approximate the flux F and its derivative inverse $G = [F']^{-1}$ for the shock and the rarefaction cases. The F is approximated by a multilayer perceptron (MLP) $\mathcal{N}_{\mathcal{F}}$. Automatic differentiation is then applied to approximate the derivative $F' \approx \mathcal{N}'_F$ and train a new MLP $\mathcal{N}_G \approx G$. Once the ANNs are trained, they can be applied to solve any Riemann problem in the flux convexity region. The results for a Fourth-Order Polynomial Flux equation are shown as numerical experiments to validate the proposed approach. The MLP architecture is chosen after several numerical trials in approximating the flux F for each test case. The results show that the ANN-Flux produced precise results when compared with analytical (when available) or numerical alternatives.

Keywords: *Deep Learning, Conservation Law, Riemann Problem*

1 Introduction

Artificial neural networks (ANNs, [4]) have been successfully applied to solve partial differential equations, mainly after the emergence of the physics-informed neural networks (PINNs, [7]). Applications to nonlinear conservation laws are also notable, including PINNs for high-speed flows ([9]), conservative PINNs (cPINNs, [5]), and weak PINNs (wPINNs, [8]). In this work, we propose the new ANN-Flux method to solve the Riemann problem for the nonlinear scalar conservation law

$$u_t + (F(u))_x = 0,\ x, t \in \mathbb{R} \times (0, t_f], \tag{1}$$

$$u(x,0) = u_L,\ x \in \mathbb{R}_-,\ u(x,0) = u_R,\ x \in \mathbb{R}_+, \tag{2}$$

with given left and right states $u_L, u_R \in \mathbb{R}$, and flux function $F : \mathbb{R} \to \mathbb{R}$.

It is a well-known fact that the problem (1) with a convex flux function has an entropic solution that is either a shock or a rarefaction wave depending on the left and right states. The exact solution of the Riemann problem for the shock case ($u_R < u_L$) is

$$u(x,t) = \begin{cases} u_L & , x < t\sigma, \\ u_R & , x > t\sigma, \end{cases} \tag{3}$$

[*]Master research fellow CAPES

and for the rarefaction wave case ($u_L < u_R$) is

$$u(x,t) = \begin{cases} u_L & ,x < tF'(u_L), \\ G(x/t) & ,tF'(u_L) < x < tF'(u_R), \\ u_R & ,x > tF'(u_R), \end{cases} \tag{4}$$

where $G(u) = [F']^{-1}(u)$, and σ is the shock speed satisfying the Rankine-Hugoniot condition $\sigma = \frac{F(u_L)-F(u_R)}{u_L-u_R}$.

The main objective of this work is to test the ANN-flux method for cases where there are two inflection points, dealing with the difficulties arising from changes in concavity and non global convexity of the selected flux.

2 ANN-Flux Method

The ANN-Flux method is designed for complicated fluxes, which are costly to evaluate, differentiate, or invert. Given that multilayer perceptrons (MLPs, [4]) neural networks are universal approximators, the ANN-Flux consists of approximating the flux F by a MLP $\tilde{y} = \mathcal{N}_F(u)$ classically trained to minimize the mean squared error (MSE) loss $\varepsilon = \varepsilon\left(\tilde{y}^{(s)}, y^{(s)}\right)$ with a generated data set $\left\{\left(u^{(s)}, y^{(s)} = F\left(u^{(s)}\right)\right)\right\}_{s=1}^{n_{s,F}}$, where $n_{s,F}$ is a given number of samples. For the simple shock wave case, the solution is then approximated by substituting F by $\mathcal{N}_F$ in (3). But, for a rarefaction case, a second MLP neural network $\mathcal{N}_G$ learns to approximate $[\mathcal{N}'_F]^{-1} \approx [F']^{-1}$. The evaluation of $\mathcal{N}'_F$ can be efficiently computed by automatic differentiation (AD, [2]). The $\mathcal{N}_G$ is trained using a directly generated data set $\left\{\left(\delta\tilde{y}^{(s)}, u^{(s)}\right)\right\}_{s=1}^{n_{s,G}}$, where $n_{s,G}$ is a given number of samples. The data $\left\{u_L \le u^{(s)} \le u_R\right\}_{s=1}^{n_{s,G}}$ is randomly generated, forward through N_F to give $\tilde{y}^{(s)} = N_F\left(u^{(s)}\right)$, and then backward propagated to compute $\delta\tilde{y}^{(s)} = \mathcal{N}'_F\left(u^{(s)}\right)$. The training of $\tilde{u}^{(s)} = \mathcal{N}_G\left(\delta\tilde{y}^{(s)}\right)$ is obtained by minimizing the MSE loss $\varepsilon = \varepsilon\left(\tilde{u}^{(s)}, u^{(s)}\right)$ on the neural network weights and biases. The solution of the rarefaction case is then given by substituting F' by $\mathcal{N}'_F$ and G by $\mathcal{N}_G$ in (4). In summary, the ANN-Flux solution of the conservation law is given by (left: shock; right:rarefaction)

$$u(x,t) = \begin{cases} u_L & ,x < t\sigma, \\ u_R & ,x > t\sigma, \end{cases} \qquad u(x,t) = \begin{cases} u_L & ,x < t\mathcal{N}'_F(u_L), \\ \mathcal{N}_G(x/t) & ,t\mathcal{N}'_F(u_L) < x < t\mathcal{N}'_F(u_R), \\ u_R & ,x > t\mathcal{N}'_F(u_R), \end{cases} \tag{5}$$

where $\sigma = \frac{\mathcal{N}_F(u_L)-\mathcal{N}_F(u_R)}{u_L-u_R}$ is Rankine-Hugoniot shock speed .

3 Results

For all the following results the neural networks have been trained with the Adam optimizer, the learning rate $l_r = 10^{-3}$ and the tolerance of $\tau = 5 \cdot 10^{-7}$ for the loss function, with hyperbolic tangent and the identity as activation functions at hidden and output layers, respectively.

Since the inverse of the flux derivative $G(u)$ is used to determine the solution in the rarefaction case, it is necessary that $F'(u)$ be monotonic to guarantee the existence of its inverse. Therefore, the neural network must be trained within the monotonic regions of $F'(u)$, so that each individual $\mathcal{N}_G$ behaves as a function.

Choosing the appropriate architecture for an MLP is done through numerical analysis. In this procedure, several architectures are tested, varying the number of hidden layers (n_h) and neurons (n_n), the network is trained and it is evaluated whether the stopping criterion, based on error or tolerance, is met, as well as the number of epochs required to reach this limit. At the end of this evaluation stage, the most efficient architecture is selected, that is, the one for which the stopping condition is satisfied using the fewest epochs.

3.1 Fourth-Order Polynomial Flux

This example was inspired by the analysis presented in [3], which details the process of constructing the Riemann solution for a nonlinear conservation laws with nonconvex flux, a fourth-order polynomial flux given by

$$F(u) = u^4 + au^3 + bu^2 + cu, \tag{6}$$

with coefficients $a = 3$, $b = -35$, and $c = -250$, and defined in the interval $u \in [-7, 7]$. The methodology developed in [3] was then expanded and applied to solve the isothermal Van der Waals gas equations.

The flux $F(u)$ exhibits two changes of concavity, which implies the existence of three distinct monotonic regions in its derivative $F'(u)$. Thus, three distinct neural networks $\mathcal{N}_G$ are needed to represent the inverse of the derivative of the flux. During the training of the MLPs, a rescaling of both inputs and outputs was required, primarily because the large magnitudes of the physical variables involved. This normalization ensures a faster training and a more precise approximation.

To choose the ANN architecture, we performed numerical tests , from these we concluded that an MLP of architecture $1-80\times5-1$ (one input, 80 units per 5 hidden layer, and one output) is adequate to approximate the flux in Table 1 with a L_2-error of 6.8×10^{-4} in fewer epochs. For the $\mathcal{N}_G^s$ the architecture chosen for $\mathcal{N}_G^0$, $\mathcal{N}_G^1$ and $\mathcal{N}_G^2$ respectively were $1-30\times5-1$, $1-30\times5-1$ and $1-40\times4-1$ found in Table 2 3 4. Figure 1 shows the comparison between ANN-Flux and the numerical solutions for all Riemann problem cases seen in [3] for the polynomial flux.

Table 1: Choosing the architecture for the neural network $\mathcal{N}_F$ for the polynomial flux. The results are n_e/ϵ, where n_e is the number of epochs and ϵ is the relative error using L_2 norm.

$n_n \backslash n_h$	3	4	5	6
50	83700/1.2E−3	75713/7.7E−4	81277/8.0E−4	81020/2.8E−3
60	56882/8.7E−4	60617/7.6E−4	58563/9.1E−4	80218/1.1E−3
70	58493/7.8E−4	63346/1.6E−3	67129/8.1E−4	72079/9.1E−4
80	50023/7.5E−4	51815/8.6E−4	41072/6.8E−4	76804/8.8E−4

Table 2: Choosing the architecture for the neural network $\mathcal{N}_G^0$. The results are n_e/ε, where n_e is the number of epochs and ε is the loss.

$n_n \backslash n_h$	2	3	4	5
30	82382/4.8E−7	48532/4.6E−7	34324/4.5E−7	24820/4.3E−7
40	72844/4.6E−7	52391/4.8E−7	25862/4.9E−7	40042/4.4E−7
50	67954/4.7E−7	37149/4.4E−7	28445/4.6E−7	28301/4.8E−7
60	67208/4.1E−7	31674/4.6E−7	28915/4.3E−7	26277/4.5E−7

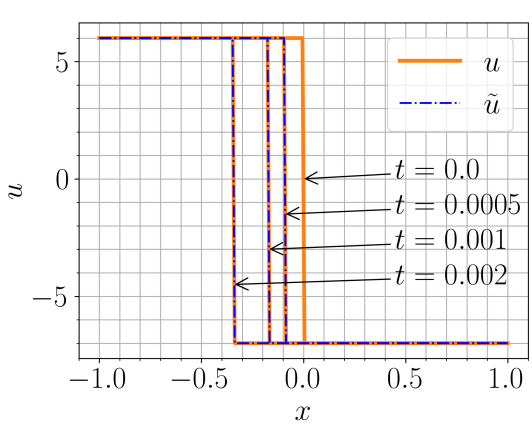
Table 3: Choosing the architecture for the neural network $\mathcal{N}_G^1$. The results are n_e/ε, where n_e is the number of epochs and ε is the loss.

$n_n \backslash n_h$	2	3	4	5
30	99999/9.5E−6	99999/7.5E−6	50238/4.4E−7	40158/4.7E−7
40	99999/9.6E−6	99999/3.5E−6	54915/4.0E−7	43942/4.5E−7
50	99999/9.2E−6	84913/1.8E−6	40322/4.3E−7	50176/4.8E−7
60	99999/1.2E−5	77503/4.4E−7	50115/4.4E−7	82491/9.1E−5

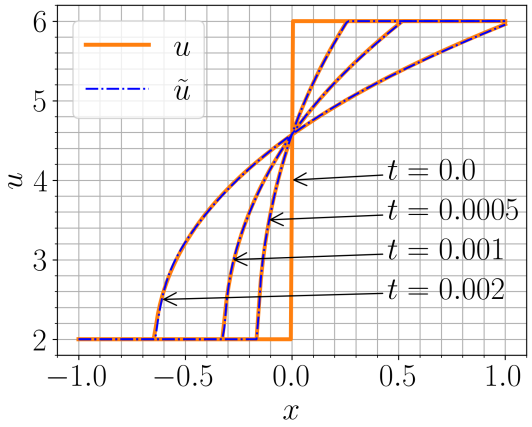
Table 4: Choosing the architecture for the neural network $\mathcal{N}_G^2$. The results are n_e/ε, where n_e is the number of epochs and ε is the loss.

$n_n \backslash n_h$	2	3	4	5
30	41588/4.8E−7	23941/4.7E−7	22111/4.7E−7	18535/4.3E−7
40	44493/4.9E−7	20589/4.6E−7	13464/4.7E−7	23035/4.8E−7
50	42597/4.6E−7	27591/4.6E−7	21490/4.2E−7	18197/4.7E−7
60	53418/4.4E−7	19836/4.3E−7	18275/4.8E−7	28638/4.8E−7

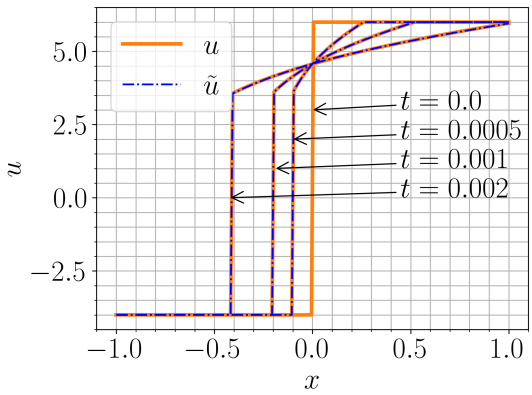
Comparisons between the ANN-Flux solution and the reference numerical solution are shown in Figure 1. For all six cases shown, the relative error between the ANN-Flux solution and the reference numerical solution remained consistently of order 10^{-4}.



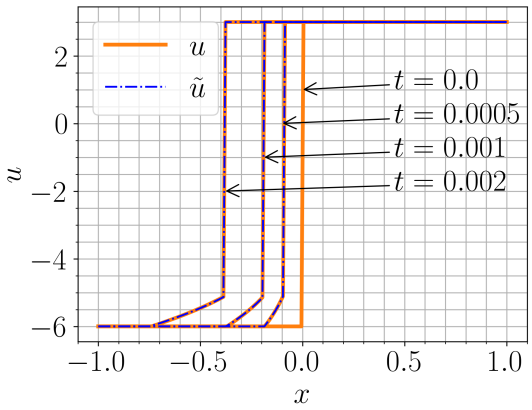
(a) Shock: $u_L = 6$, $u_R = -7$



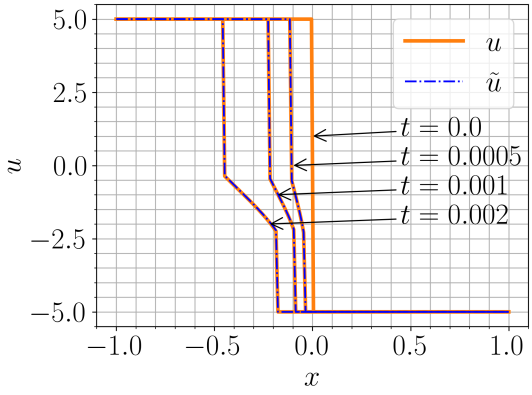
(b) Rarefaction: $u_L = 2$, $u_R = 6$



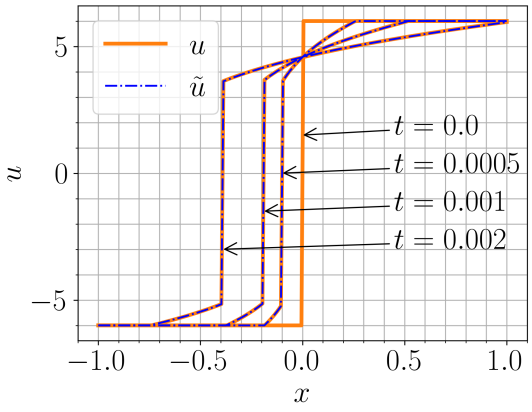
(c) Shock-Rarefaction: $u_L = -4$, $u_R = 6$



(d) Rarefaction-Shock: $u_L = -6$, $u_R = 3$



(e) Shock-Rarefaction-Shock: $u_L = 5$, $u_R = -5$



(f) Rarefaction-Shock-Rarefaction: $u_L = -6$, $u_R = 6$

Figure 1: ANN-Flux ($\tilde{u}$) *versus* numerical solutions (u) of the Polynomial Flux Equation. Source: From Author.

4 Concluding Remarks

The results demonstrate that the ANN-Flux method is effective in approximating the flux function F and its derivative inverse $G = [F']^{-1}$ in the fourth-order polynomial flux equation, This approach shows great promise for solving nonlinear conservation laws where approximating a nonconvex flux functions may be computationally expensive or infeasible. Once the required ANNs are trained, the ANN-Flux method provides precise solution structures, including shocks, rarefactions, and combinations thereof. Future directions include extending the method to systems of conservation laws and developing adaptive strategies for training the neural networks on-the-fly during simulation.

5 Bibliografy

References

[1] P. Amorim, R. Colombo e A. Teixeira, "On the numerical integration of scalar nonlocal conservation laws", *ESAIM: Mathematical Modelling and Numerical Analysis*, **49**(1), 19–37 (2015). DOI: 10.1051/m2an/2014023.

[2] A. G. Baydin, B. A. Pearlmutter, A. A. Radul e J. M. Siskind, "Automatic Differentiation in Machine Learning: a Survey", *Journal of Machine Learning Research*, **18**(153), 1–43 (2018).

[3] M. Fossati e L. Quartapelle, "The Riemann Problem for Hyperbolic Equations under a Nonconvex Flux with Two Inflection Points", , arXiv preprint *arXiv:1402.5906*, (2014). Disponï¿½vel em: `https://arxiv.org/abs/1402.5906`.

[4] S. Haykin, "Neural Networks and Learning Machines", 3a. ed., Pearson, New York, 2009.

[5] A. D. Jagtap, E. Kharazmi e G. E. Karniadakis, "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems", *Computer Methods in Applied Mechanics and Engineering*, **365**, 113028 (2020). DOI: 10.1016/j.cma.2020.113028.

[6] R. J. LeVeque, "Numerical Methods for Conservation Laws", Springer, 1992.

[7] M. Raissi, P. Perdikaris e G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations", *Journal of Computational Physics*, **378**, 686–707 (2019). DOI: 10.1016/j.jcp.2018.10.045.

[8] T. Ryck, S. Mishra e R. Molinaro, "wPINNs: Weak Physics Informed Neural Networks for Approximating Entropy Solutions of Hyperbolic Conservation Laws", arXiv preprint *arXiv:2207.08483*, (2022). DOI: 10.48550/arXiv.2207.08483.

[9] Z. Mao, A. D. Jagtap e G. E. Karniadakis, "Physics-informed neural networks for high-speed flows", *Computer Methods in Applied Mechanics and Engineering*, **360**, 112789 (2020). DOI: 10.1016/j.cma.2019.112789.