The Influence of the Optimization Algorithm in the Solution of the Fractional Laplacian Equation by Neural Networks

C. Coelho[1,a)], M. Fernanda P. Costa[1,b)] and L.L. Ferrás[1,c)]

[1]*Centre of Mathematics, University of Minho, 4710 - 057 Braga, Portugal*

[a)]Corresponding author: ceciliaeduarda58@gmail.com
[b)]mfc@math.uminho.pt
[c)]luislimafr@gmail.com

Abstract. In this paper, the influence of the optimization algorithms Adam, RMSprop, L-BFGS and SGD with momentum on the solution of Fractional Laplacian Equation (FLE) by physics-informed neural networks is analysed when considering two different analytical solutions, one smooth and one non-smooth. The influence of the optimization method, the smoothness of the analytical solution and the network configuration on the accuracy of the predicted solution is discussed in detail.

INTRODUCTION

Physics-informed neural networks (PINN) [1, 2] are neural networks that are trained to solve supervised learning tasks while respecting any given laws of physics described by general nonlinear partial differential equations and fractional differential equations (integral operators) - fPINN [3].

In this work we aim to analyse the effect of different optimization algorithms when we use fPINN to solve the following 1D fractional Laplacian equation (FLE) with the boundary conditions:

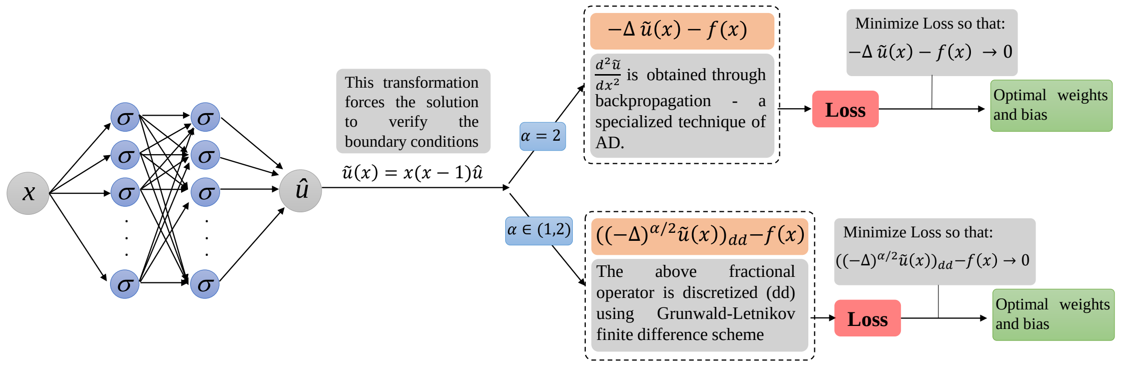
$$(-\Delta)^{\alpha/2}u(x) = f(x), \quad u(0) = u(1) = 0, \quad x \in (0,1),\ \alpha \in (1,2) \tag{1}$$

where the fractional Laplacian operator $(-\Delta)^{\alpha/2}$ is given by:

$$(-\Delta)^{\alpha/2}u(x) = \frac{1}{2\cos\left(\frac{\alpha\pi}{2}\right)}\left(\frac{1}{\Gamma(n-\alpha)}\frac{d^n}{dx^n}\left(\int_{0+}^{x}\frac{u(s)}{(x-s)^{\alpha-n+1}}ds\right) + \frac{(-1)^n}{\Gamma(n-\alpha)}\frac{d^n}{dx^n}\left(\int_{x}^{1-}\frac{u(s)}{(s-x)^{\alpha-n+1}}ds\right)\right), \tag{2}$$

with $n = [\alpha] + 1$ and $[\alpha]$ the integer part of α. The 1D FLE finds application, for example, in electrostatics and anomalous concentrations. To solve the 1D FLE using fPINN, we follow the schematic shown in Figure 1. For the case of classical partial differential equations (PDE), automatic differentiation (AD) is used to obtain an analytical expression that mimics the original PDE. This analytical expression is then used to compute the loss function [3]. Minimization of the loss function of fPINN [3] is performed over the training dataset $\{x_i : x_i \in (0,1),\ \ i = 1,...,N\}$, being this minimization process called training. At the end of the training process, we expect to obtain a set of optimal weights and bias that allow our model to predict a good solution for each point in the testing dataset $\{x_j : x_j \in (0,1),\ \ j = 1,...,M\}$ where each $x_j \neq x_i$ for $i = 1,\dots,N$. For the case of fractional differential equations, the classical chain rule is no longer valid and consequently AD can not be used. Therefore, the analytic expressions are obtained by a discretization of the original operators (e.g., a finite integral can be approximated by a finite sum). In this work, we use the Grünwald-Letnikov finite difference scheme. In addition to the training and testing datasets, another dataset of points (known as auxiliary points) is built that help to compute the fractional derivatives.

The aim of this work is then to investigate the influence of the optimization algorithms: Adam, RMSprop, L-BFGS, SGD with momentum, the smoothness of the analytical solution and the neurons distribution, on the accuracy of the predicted solution. The numerical tests are performed using the DeepXDE library [1].



x
σ
$\hat{u}$
This transformation forces the solution to verify the boundary conditions
$\tilde{u}(x) = x(x-1)\hat{u}$
$\alpha = 2$
$\alpha \in (1,2)$
$-\Delta\,\tilde{u}(x) - f(x)$
$\frac{d^2\tilde{u}}{dx^2}$ is obtained through backpropagation - a specialized technique of AD.
$((-\Delta)^{\alpha/2}\tilde{u}(x))_{dd} - f(x)$
The above fractional operator is discretized (dd) using Grunwald-Letnikov finite difference scheme
Loss
Minimize Loss so that: $-\Delta\,\tilde{u}(x) - f(x) \to 0$
Minimize Loss so that: $((-\Delta)^{\alpha/2}\tilde{u}(x))_{dd} - f(x) \to 0$
Optimal weights and bias

FIGURE 1. Schematic of the methodology used in the PINN method for the solution of differential equations using classical and fractional (integral) operators (fPINN). Since the boundary conditions are $u(0) = u(1) = 0$, we simply choose the surrogate model $\tilde{u}(x) = x(x-1)\hat{u}(x)$ to satisfy the boundary condition automatically, where $\hat{u}(x)$ is the output of the neural network.

RESULTS and DISCUSSION

We will now analyse the performance of the optimization algorithms for both smooth and non-smooth solutions, with a learning rate of 10^{-3} and a maximum of 10000 steps (default settings). Note that Adam and RMSprop are adaptive learning rate methods and L-BFGS computes the step size by using the Wolfe conditions. We consider two different feed-forward NN. One with 5 hidden layers of 50 neurons each 50-50-50-50-50 (denoted by *constant* NN (c)), and another with also 5 hidden layers but with a 128-64-32-16-8 neurons distribution (denoted by *decreasing* NN (d)).

Smooth Solution and $\alpha = 1.99$

We start the study by considering the numerical solution of (1)-(2) with $f(x) = \left(2\cos\left(\frac{\pi\alpha}{2}\right)\right)^{-1} \times \left[\frac{\Gamma(4)}{\Gamma(4-\alpha)}(x^{3-\alpha} + (1-x)^{3-\alpha}) - \frac{3\Gamma(5)}{\Gamma(5-\alpha)}(x^{4-\alpha} + (1-x)^{4-\alpha}) + \frac{3\Gamma(6)}{\Gamma(6-\alpha)}(x^{5-\alpha} + (1-x)^{4-\alpha}) - \frac{\Gamma(7)}{\Gamma(7-\alpha)}(x^{6-\alpha} + (1-x)^{6-\alpha}\right]$. The analytical solution is smooth and is given by $u(x) = x^3(1-x^3)$. The training was performed with 20 points in the domain.

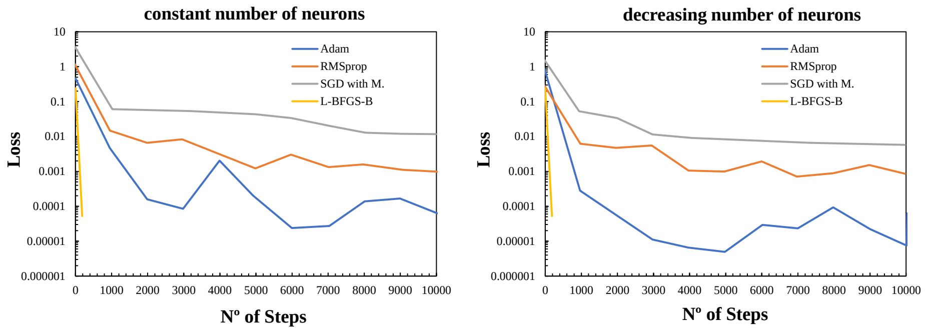


FIGURE 2. Evolution of the loss during training, for the algorithms Adam, RMSprop, L-BFGS and SGD (with momentum), and, for the two different architectures proposed (with the *constant* NN (c) and *decreasing* NN (d)). Case of a smooth solution.

TABLE 1. Performance metrics and training time obtained by solving the FLE ($\alpha = 1.99$), with the *constant* NN (c) and *decreasing* NN (d), using different optimizers.

	L2 relative error (c)	Mean square error (c)	Training time (seconds) (c)	L2 relative error (d)	Mean square error (d)	Training time (seconds) (d)
Adam	0.026	5.422e-8	16.926	0.002	2.103e-10	17.740
RMSprop	0.029	6.416e-8	18.790	0.090	6.682e-7	18.405
L-BFGS	0.001	9.998e-11	5.585	0.001	1.344e-10	4.162
SGD (with momentum)	0.033	9.717e-8	17.178	0.017	2.373e-8	17.763

Figure 2 shows the evolution of the loss during training, for the algorithms Adam, RMSprop, L-BFGS and SGD (with momentum), generally obtaining a decreasing loss. The best results are obtained for Adam and L-BFGS, the latter being notable for the steep reduction in the loss function. We note that the L-BFGS has a stopping criteria not only based on the maximum number of steps and consequently reaches its optimality criteria before reaching 10000 steps. The worst results are obtained by SGD (with momentum) and RMSprop, showing the difficulty to effectively reduce the training loss. The *decreasing* NN allows Adam to further reduce the loss function (compared to the *constant* NN). The differences are less pronounced for the other algorithms. Moreover, L-BFGS is able to achieve the lowest loss value of all methods, with the *constant* NN architecture.

These results are summarised in Table 1, where the $L2$ relative error, the mean square error and the training time (in seconds) for the two neural networks, *constant* NN (c) and *decreasing* NN (d), are presented.

The L-BFGS algorithm has the lowest loss value and lower computation time. However, we cannot compare the training time of this method with the others because the stopping criteria are not comparable. Nevertheless, L-BFGS achieves the lowest loss function value among all methods when considering the same number of steps, and yields the smaller $L2$ relative and mean square errors.

Non-Smooth Solution and $\alpha = 1.11$

We now consider the numerical solution of of (1)-(2) with $f(x) = \left(2\cos\left(\frac{\pi\alpha}{2}\right)\right)\Gamma(\alpha+2)x$. The analytical solution is non-smooth, $u(x) = x(1-x^2)^{\alpha/2}$. This case is more extreme, and is expected to bring increased difficulties in the approximation of the fractional operator (by the difference scheme) and in the solution by fPINN.

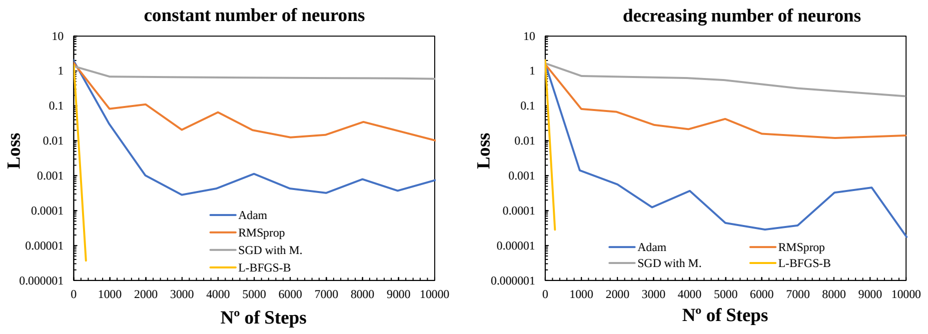


FIGURE 3. Evolution of the loss during training, for Adam, RMSprop, L-BFGS and SGD (with momentum) algorithms, and, for the two different architectures proposed (with the *constant* NN (c) and *decreasing* NN (d)). Case of a non-smooth solution.

TABLE 2. Performance metrics and training time obtained by solving the FLE ($\alpha = 1.11$), with *constant* NN (c) and *decreasing* NN (d), using different optimizers.

	L2 relative error (c)	**Mean square error (c)**	**Training time (seconds) (c)**	**L2 relative error (d)**	**Mean square error (d)**	**Training time (seconds) (d)**
Adam	0.102	0.001	15.967	0.102	0.001	16.914
RMSprop	0.104	0.001	16.646	0.099	0.001	16.132
L-BFGS	0.100	0.001	26.438	0.102	0.001	13.631
SGD (with momentum)	0.267	0.009	15.871	0.193	0.005	15.313

Figure 3 shows the evolution of the loss during training for the two different proposed architectures: *constant* NN (c) and *decreasing* NN (d). As expected, the values of the loss function are higher compared to the smooth solution case, and the algorithms have difficulty in reducing the loss along the steps. Note that SGD (with momentum) hardly reduces the loss in 10000 steps, behaving slightly better with the *decreasing* NN. The results show significant improvement for the Adam algorithm when using the *decreasing* NN, managing to reach the lowest loss value of all the experiments performed for the non-smooth solution.

Table 2 shows the $L2$ relative error, the mean square error and the training time (in seconds) for each optimization algorithm and for the two neural networks, *constant* NN (c) and *decreasing* NN (d). We see that the errors are quite high and fairly worse than the results achieved for the smooth solution. The SGD (with momentum) optimizer presents the highest errors while the other three methods have similar performances, having an $L2$ error of ≈ 0.1 and a mean square error of ≈ 0.001. These results were expected due to the singularity at $x = 1$. The high gradients near the boundary lead to an increased difficulty for the model to predict accurate solutions (as shown in Figure 4 for the algorithms L-BFGS and SGD). In terms of computational time, there are no significant differences except for L-BFGS which did not keep the fast convergence behaviour (in the smooth solution), although performing less steps than the other methods. This demonstrates that the computational costs of using a Hessian approximation are higher in this case (non-smooth solution).

Figure 4 shows that more points would be needed near $x = 1$ to capture the high gradients. Note that the error arises from both the difference scheme used to approximate the fractional operator and from the network.

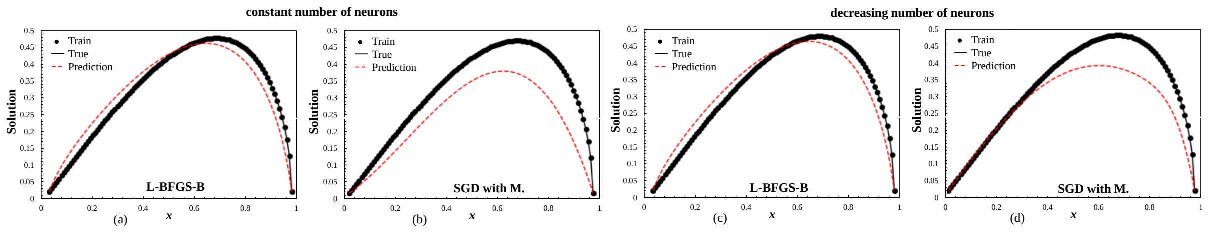


FIGURE 4. True solution vs prediction for the algorithms that presented the smaller (L-BFGS) and higher (SGD with momentum) error: (a) and (b) *constant* NN; (c) and (d) *decreasing* NN.

We increased the number of points in the training dataset and also the number of auxiliary points, but the model improved only slightly, keeping the $L2$ errors on the order of ≈ 0.1.

CONCLUSIONS

The influence of the optimization algorithms Adam, RMSprop, L-BFGS and SGD with Momentum on the solution of FLE by physically informed neural networks was analysed. These algorithms were tested in two different neural network configurations, and considering smooth and non-smooth solutions.

The L-BFGS algorithm was found to be the most efficient algorithm, providing the most accurate solutions for both networks and type of solution. However, for the non-smooth solutions, the high computational cost can be prohibitive, making Adam a good alternative. As expected, SGD (with momentum) shows the worst performance overall, which may be explained by the constant learning rate (10^{-3}) throughout the training. In contrast, at each step, Adam and RMSprop computes adaptive learning rates for each parameter. The L-BFGS algorithm, at each step, computes the learning rate based on the Wolfe conditions. For the case of non-smooth solution, a curious remark is that by observing the variation of the loss function, we can see that L-BFGS and Adam achieve the same values, although Adam requires more steps but not necessarily more time. This may be explained by the fact that L-BFGS performs updates at each iteration using a search direction that requires the limited-memory inverse Hessian approximation, while Adam only uses a search direction computed based on past gradients. Note that the *decreasing NN* architecture allows the L-BFGS algorithm to decrease its computational time. Furthermore, none of the optimizers were able to provide highly accurate solutions for strong singular solutions. In the future, we will test different configurations and distributions of points in the training and auxiliary datasets to better capture the solution near the singularity.

ACKNOWLEDGMENTS

The authors acknowledge the funding by Fundação para a Ciência e Tecnologia (Portuguese Foundation for Science and Technology) through projects UIDB/00013/2020 and UIDP/00013/2020.

REFERENCES

[1] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, SIAM Review **63**, 208–228 (2021).
[2] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Journal of Computational Physics **378**, 686–707February (2019).
[3] G. Pang, L. Lu, and G. E. Karniadakis, SIAM Journal on Scientific Computing **41**, A2603–A2626 (2019), https://doi.org/10.1137/18M1229845 .