OPT2020: 12th Annual Workshop on Optimization for Machine Learning

Shallow Physics Informed Neural Networks Using Levenberg-Marquardt Optimization

Gaurav Kumar Yadav ME16D022@SMAIL.IITM.AC.IN
Balaji Srinivasan SBALAJI@IITM.AC.IN
IIT Madras, Chennai, India

Abstract

There is a renewed interest in exploring the application of Artificial Neural Networks (ANNs) to solve Differential equations. One of the popular methods is Physics informed Neural Networks(PINNs), which embeds the knowledge of the equation itself into the loss function of the Neural-Network. The traditional PINNs use Multi-layer ANNs (MLNNs) and employ Gradient-Descent type optimization algorithms like Adam/L-BFGS-B to optimize the weights of the ANN. In this paper, we explore the well-known Levenberg-Marquardt (LM) Optimization algorithm to optimize the weights of a Single-layer Neural Network (SLNN) based PINNs. We show that for a class of problems known as Singular Perturbation Problems (SPPs), our method can achieve much more accurate solutions, much faster, than the Traditional PINNs. The prevalent research on ANNs mostly focuses on the architecture and the data. Based on our observations, we establish that the choice of weight optimization algorithms are as important as the other two and need due consideration.

1. Introduction

Artificial Neural Networks (ANNs) are capable of fitting highly non-linear data, owing to their superior representational capability. Differential equations govern most physical phenomena, and their solutions involve non-linear relations between independent and dependent variables; this makes ANNs a viable choice to seek the solutions of PDEs/ODEs.

Lagaris et al. [6] first proposed incorporating the Differential equation itself along with it's Boundary/Initial conditions into the Loss function of the ANNs. Recently Raissi et al. [9], used a similar approach, called Physics informed Neural-Networks (PINNs) to solve linear and non-linear problems. Dwivedi and Srinivasan [1], devised a method which is much more efficient for linear problems, and uses a shallow single-layer network. Further works by Jagtap et al. [4], Dwivedi et al. [2], decomposed the solution domain of the PDEs into smaller parts to address problems where the solutions of PDEs or ODEs were non-smooth, by employing a Deep ANNs into each sub-domain.

All these methods, except PIELM, by Dwivedi and Srinivasan [1], either use Deep Neural-Networks with Multiple Neurons or use Multiple Shallow Networks in small sub-domains. To optimize the weights of these Mutil-layer PINNs, these methods use Gradient-Descent like algorithms, like Adam, or L-BFGS-B or their combination. The fact that there are a large number of weights to be optimized leads to slow convergence and high computational cost.

It is a well-known fact that when the number of parameters to be optimized is moderate, the LM, invented by Levenberg [7] and Marquardt [8], is the most efficient Optimization algorithm [3] . Exploiting this fact, we propose SLNN-PINN with a small number of neurons in the hidden layer,

to solve a class of problems called SPPs. We show that for this category of problems, SLNN-PINN combined with the LM, converges much faster than Multilayer-PINNs combined with Adam/L-BFGS-B, and provides much better accuracy.

We start by giving a brief problem statement, followed by the implementation of SLNN-PINN with LM. We then show some of the essential results, and finally, conclude by highlighting our essential findings.

2. Singular Perturbation Problems

SPPs are differential equations, with applications in flow modeling near the bodies' surfaces, such as in Aerodynamics, Heat-Exchangers, etc. Their solutions have a characteristic presence of thin regions where the function that is the solution has a sharp gradient. A model 1-D SPP, the kind that we have solved in this paper, looks like the following:[5]

$$\epsilon \frac{d^2u}{dx^2} + a(x)\frac{du}{dx} + b(x)u = f(x); \qquad x \in [0,1], \qquad \epsilon \ll 1 \tag{1}$$

With Boundary conditions,

$$u(0) = \alpha \quad \text{and} \quad u(1) = \beta \tag{2}$$

3. Methodology

We describe here the formulation of SLNN based PINNs, followed by the Optimization algorithm.

3.1. SLNN based PINN Formulation

We shown the formulation here by taking the SPP 1, along with the Boundary conditions 2, though the Formulation can be extended to any general differential equation.

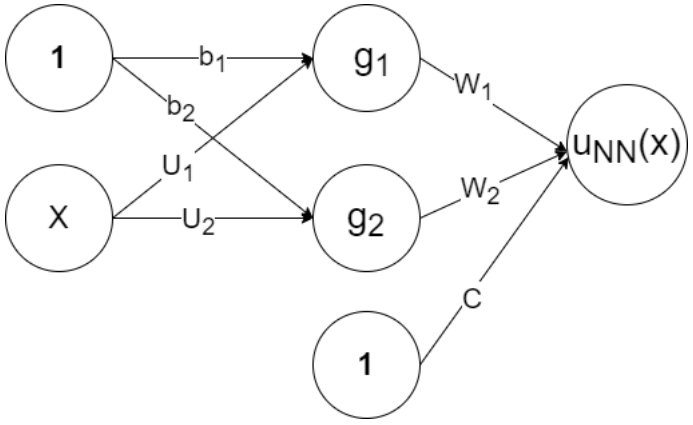


Figure 1: SLNN used in the formulation, shown with 2 nodes

For any given data point x, the output of the i^{th} neuron in the hidden layer is given by

$$g_i(x) = tanh(U_i x + b_i) \tag{3}$$

The Output of the ANN is,

$$u_{NN}(x) = \sum_{i=1}^{N}(W_i g_i(x)) + C; \quad : N = \text{number of Neurons} \tag{4}$$

Then, the derivatives are obtained by,

$$\frac{du_{NN}}{dx}|_x = \sum_{i=1}^{N} W_i U_i (1 - g_i^2) \quad \text{and} \quad \frac{d^2u_{NN}}{dx^2}|_x = -\sum_{i=1}^{N} 2 g_i W_i U_i^2 (1 - g_i^2) \tag{5}$$

Error at any internal domain point can then be calculated as,

$$D_E(x) = 0 - (\epsilon \frac{d^2u_{NN}}{dx^2} + a(x)\frac{du_{NN}}{dx} + b(x)u_{NN} - f(x)) \tag{6}$$

Errors at the boundaries can be evaluated by,

$$V_E(0) = \alpha - (\sum_{i=1}^{N}(W_i g_i(0)) + C) \quad \text{and} \quad V_E(1) = \beta - (\sum_{i=1}^{N}(W_i g_i(1)) + C) \tag{7}$$

Then, the total error can be formulated as,

$$TSE = V_E(x_L)^2 + \sum_x D_E(x)^2 + V_E(x_R)^2 \quad \text{or} \quad TSE = \mathbf{f}^T\mathbf{f} \tag{8}$$

$$: \mathbf{f}(\mathbf{x}, \mathbf{U}, \mathbf{b}, \mathbf{W}, C) = [V_E(x_L) \;\; D_E(x_1) \;\; D_E(x_2) \;\; ....D_E(x_m) \;\; V_E(x_R)] \tag{9}$$

let $\mathbf{P} = [\mathbf{U} \;\; \mathbf{b} \;\; \mathbf{W} \;\; C]^T$.
The Jacobian matrix will be,

$$J_{ij} = \frac{\partial f_i}{\partial P_j} \tag{10}$$

: i goes from 1 to $m + 2$
: j goes from 1 to $3N + 1$
: m is the number of collocation data points, where the errors in the differential equation are evaluated.
: N is the number of neurons in the hidden layer.

Our goal is to find that $\mathbf{P}$, which will minimize the TSE. To achieve that, we start with some initial guess for $\mathbf{P}$ and then improve upon it by using the method below:

$$\mathbf{P}_{new} = \mathbf{P}_{old} - (J^T J + \lambda I)|_{old}^{-1}\mathbf{f}|_{old} \tag{11}$$

Note that the TSE must be properly scaled. For the results in this paper, we scaled TSE by multiplying it by m^4. Our idea is that scaling TSE leads to better-conditioned J.

3.2. Optimization using LM

Levenberg-Marquardt Optimization algorithm incorporates properties of both Steepest-Descent and Newton's method. Below is the algorithm for SLNN based PINNs. MATLAB®'s $lsqnonlin$ function was used in the simulations.

Algorithm 1: Levenberg-Marquardt Optimization Algorithm

Input: Data $\mathbf{x} \in \mathbb{R}^m$, Initial Guess for weight vector $\mathbf{P}$, $\nu > 1$, Convergence criteria $\mu \in \mathbb{R}_{>0}$, $\lambda = 0.01$

Compute TSE using 8

while $TSE > \mu$ **do**

  Compute $\mathbf{f}$ and J using 9 and 10

  Compute $\mathbf{P}^*$ as $\mathbf{P}_{new}$ using 11 taking current values of $\mathbf{f}$, J and λ as old values.

  Compute TSE^* using $\mathbf{P}^*$.

  if $TSE^* < TSE$ **then**

    $TSE := TSE^*$

    $\mathbf{P} := \mathbf{P}^*$

    $\lambda := \frac{\lambda}{\nu}$

  else

    while $TSE^* > TSE$ **do**

      $\lambda := \nu\lambda$

      Compute $\mathbf{P}^*$ as $\mathbf{P}_{new}$ using 11 taking current values of $\mathbf{f}$, J and λ as old values.

      Compute TSE^* using $\mathbf{P}^*$.

    end

    $TSE := TSE^*$

    $\mathbf{P} := \mathbf{P}^*$

  end

end

If λ is not decreasing, then the pseudo-hessian $J^T J$, causes LM's convergence to be like Steepest-Descent; this is preferable when we are far from the optimum. As λ starts decreasing and becomes small, LM switches to Quadratic convergence, like Newton's method, this happens when we are near the optimum. Detailed analysis can be found in [7, 8, 10]

4. Results

In this section, we apply our methods on two SPPs. For case 1, we compare multi-layer PINNs using the combination of Adam and L-BFGS-B as optimizer to our single layer PINN using LM.

4.1. Case 1

Here we solve the an SPP which does not have a reaction term.

$$\epsilon\frac{d^2u}{dx^2} + 2\frac{du}{dx} = 0; \qquad x \in [0, 1], \qquad \epsilon \ll 1 \tag{12}$$

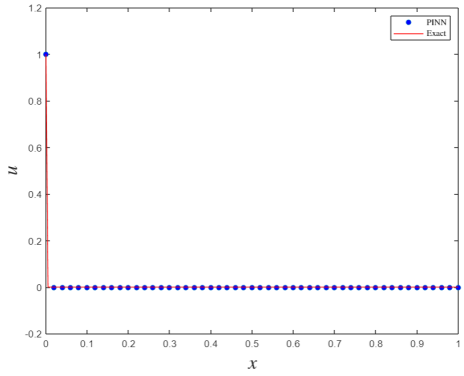
With Boundary conditions,

$$u(0) = 1.0 \quad \text{and} \quad u(1) = 0.0 \tag{13}$$

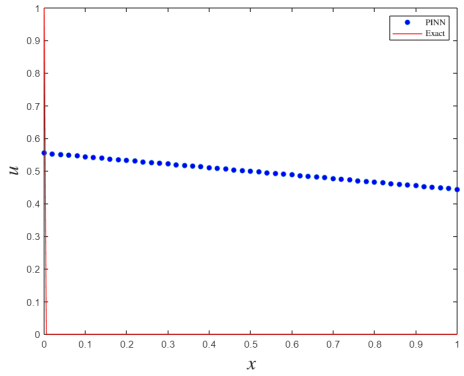
The exact solution is

$$u(x) = \frac{exp(\frac{-2x}{\epsilon}) - exp(\frac{-2}{\epsilon})}{1 - exp(\frac{-2}{\epsilon})} \tag{14}$$

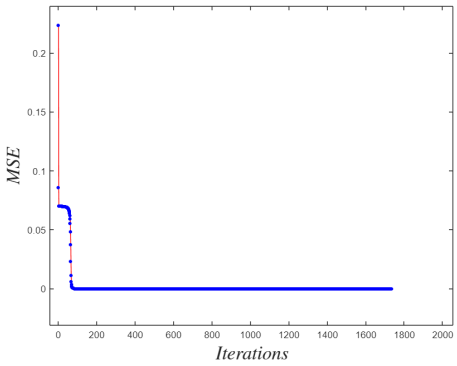
We compare the Mean Square Error (MSE), instead of TSE. The former having smaller numbers is easier to compare. The conversion formula TSE, scaled by multiplying m^4 is, $TSE = m^4 MSE$, where m is the number of data points.



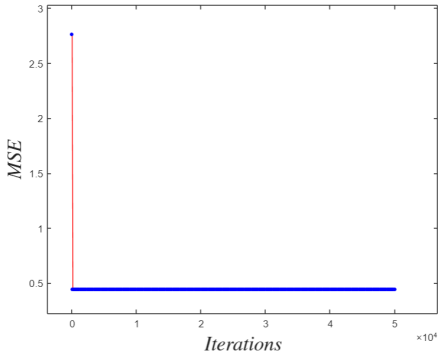
(a) Single-Layer PINN with LM



(b) Multi-Layer PINN with Adam+L-BFGS-B



(c) Single-Layer PINN with LM



(d) Multi-Layer PINN with Adam+L-BFGS-B

Figure 2: Comparison is shown here for $\epsilon = 1e-03$. The top row shows $u - x$ fit based on last iteration values of PINNs weights. The bottom row shows decrease in MSE as the iterations progress. MLNN based PINN with Adam+L-BFGS-B failed consistently over several trials, the MSE did not decrease for it below 0.44

Table 1: Performance comparison for $\epsilon = 1e-03$

Algorithm	**Layer/Neurons**	**Data-Points**	**Iterations**	**MSE**	**Platform**
LM	1/4	51	500-2000	2e-10	MATLAB®
Adam+L-BFGS-B	5/20	2001	50000+4	0.44	TensorFlow™

4.2. Case 2

Here, we apply our method on another SPP, where the reaction term is the function of the perturbation parameter ϵ.

$$-\epsilon\frac{d^2u}{dx^2}+\frac{du}{dx}+(1+\epsilon)u=0, \qquad x\in[0,1], 0<\epsilon \qquad \ll 1 \tag{15}$$

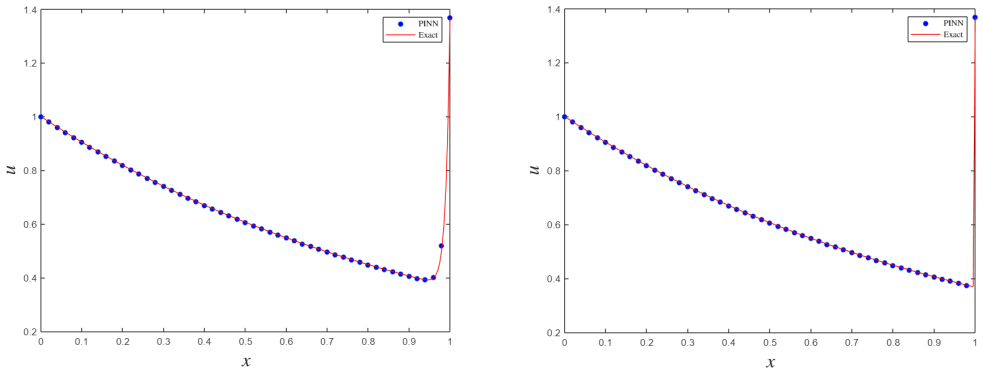
With boundary conditions given by,

$$u(x=0)=1.0+exp(-\frac{(1+\epsilon)}{\epsilon}) \quad \text{and} \quad u(x=1.0)=1.0+exp(-1) \tag{16}$$

The Exact solution is given by:

$$u(x)=exp(\frac{(1+\epsilon)(x-1)}{\epsilon})+exp(-x) \tag{17}$$

We solved this problem for many values of ϵ. We show here the result for $\epsilon = 1e-02$ and a much smaller value $\epsilon = 1e-05$.
For both these simulations, the number of data points were fixed to 45. The scaling of TSE was set to m^2 and the Number of neurons in the single-layer was fixed to 4.



(a) $u-x$ fit for $\epsilon = 1e-02$ (b) $u-x$ fit for $\epsilon = 1e-05$

Figure 3: Solution of SPP 15, with boundary conditons 16

5. Conclusion

We explored the use of LM optimization algorithm for optimizing the weights of the SLNN based PINNs. We established that for SPPs, our method gives much more accurate results, faster, MLNN based PINNs using Adam/L-BFGS-B optimizers. We believe that by using better optimization algorithms along with dividing domains of complex problems into small sub-domains, Shallow Networks might be able to compete with Deep Networks in representing complex phenomenons.

Acknowledgements

This work was supported by Robert Bosch Centre for Data Science and Artificial Intelligence, Indian Institute of Technology- Madras, Chennai (Project No. CR1920ME615RBCX008832).

References

[1] Vikas Dwivedi and Balaji Srinivasan. Physics informed extreme learning machine (pielm)–a rapid method for the numerical solution of partial differential equations. *Neurocomputing*, 391:96 – 118, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2019.12.099. URL http://www.sciencedirect.com/science/article/pii/S0925231219318144.

[2] Vikas Dwivedi, Nishant Parashar, and Balaji Srinivasan. Distributed learning machines for solving forward and inverse problems in partial differential equations. *Neurocomputing*, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2020.09.006. URL http://www.sciencedirect.com/science/article/pii/S0925231220314090.

[3] M. T. Hagan and M. B. Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993, 1994.

[4] Ameya D. Jagtap, Ehsan Kharazmi, and George Em 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. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2020.113028. URL http://www.sciencedirect.com/science/article/pii/S0045782520302127.

[5] Vivek Kumar and Balaji Srinivasan. An adaptive mesh strategy for singularly perturbed convection diffusion problems. *Applied Mathematical Modelling*, 39(7):2081 – 2091, 2015. ISSN 0307-904X. doi: https://doi.org/10.1016/j.apm.2014.10.019. URL http://www.sciencedirect.com/science/article/pii/S0307904X14004922.

[6] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[7] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *quarterly of applied mathematics*, 1944. ISSN 0033-569X. doi: https://doi.org/10.1090/qam/10666. URL https://www.ams.org/journals/qam/1944-02-02/S0033-569X-1944-10666-0/S0033-569X-1944-10666-0.pdf.

[8] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963. doi: 10.1137/0111030. URL https://doi.org/10.1137/0111030.

[9] M. Raissi, P. Perdikaris, and 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. ISSN 0021-9991.

doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL http://www.sciencedirect.com/science/article/pii/S0021999118307125.

[10] Sam Rowies. Levenberg-marquardt optimization. *Note*. URL https://cs.nyu.edu/~roweis/notes/lm.pdf.