POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

EXECUTIVE SUMMARY OF THE THESIS

Artificial Neural Neworks for the approximate solution of Partial Differential Equations

LAUREA MAGISTRALE IN MATHEMATICAL ENGINEERING - INGEGNERIA MATEMATICA

Author: BEATRICE CRIPPA

Advisor: PROF.SSA PAOLA FRANCESCA ANTONIETTI

Academic year: 2020-2021

1. Introduction

Partial Differential Equations (PDEs) are widely used to model many physical, financial and social phenomena. There exists a wide class of numerical schemes employed for their approximate solution such as Finite Element methods. Finite Element methods are based on the construction of a suitable triangulation of the domain and the definition therein of a discrete space [1]. The computational cost of FEMs grows with the domain dimension, and as a consequence incur in the so-called curse of dimensionality.
Since deep learning techniques have gained success in high-dimensional data frames, new numerical methods have been introduced in the past years for the approximate solution of PDEs, based on Artificial Neural Networks (ANNs). ANNs are learning structures based on the synaptic neural connection in the human brain, and have been proved to be able to approximate a wide set of functions [2, 4–6]. On the one hand, ANN-based methods can overcome the curse of dimensionality and provide a flexible paradigm because they are mesh-free. On the other hand, the lack of a strong theoretical background and lack of information of the physics of the underlying problem represent important drawbacks.
In my thesis I present an ANN-based method for the numerical solution of PDEs, based on the work of Xu et al. [14] and Karniadakis [9] and test its performance on a wide class of problems in terms of approximation error, stability and computational time.
This executive summary is organized as follows: Section 2 provides a brief introduction to ANNs and the state of the art on their application to PDEs, Section 3 introduces the numerical method discussed in my thesis, followed by its application to the main test cases considered in Section 4. In Section 5 some conclusions and further developments are discussed.

2. Artificial Neural Networks

ANNs are computational learning systems transforming data inputs into numeric outputs, whose building blocks are interconnected nodes, called neurons, that are grouped into layers. Their main applications are classification, pattern recognition, and prediction in many disciplines, usually providing reasonable robustness.
We consider feedforward dense neural networks, made of one input layer containing the data, one output layer providing the result and some intermediate layers called hidden.
Each neuron takes as input a vector $\mathbf{x} = [x_1, x_2, ..., x_N]$ of data whose entries coincide

with the output of all the neurons in the previous layer, and gives a 1-dimensional output.

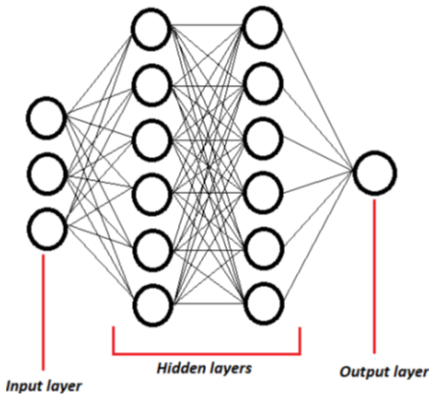

Figure 1: Structure of a feedforward dense neural network with two hidden layers made of six neurons each. The input layer consists of the input data and the output layer gives the result. The middle layers are called hidden and are made of interconnected nodes, called neurons.

Let us enumerate the neurons in each layer $l = 1, ..., L$ from 1 to N^l. Then, the output a^l_j of neuron j in layer l is computed as the evaluation of a chosen real activation function σ at the weighted sum of the neuron input with respect to specific parameters w^l_{jk}, $j = 1, ..., N^l$, $k = 1, ..., N^{l-1}$, $l = 1, ..., L$ and a bias b^l_j, $l = 1, ..., l$. The weights w^l_{jk} are specific for each connection between the k^{th} neuron in layer $l-1$ and the j^{th} neuron in layer l. It can be expressed as follows:

$$a^l_j = \sigma\left(\sum_{k=1}^{N^{l-1}} w^l_{jk} a^{l-1}_k + b^l_j\right),$$

$\forall j = 1, ..., N^l$ $\forall l = 1, ..., L$. An example of structure of a backpropagation ANN is represented in Figure 2.
The weights $\{w^l_{jk}\}_{jkl}$ and biases $\{b^l_j\}_{jl}$ are automatically chosen during the network training phase, according to the minimization via gradient descent-based algorithms of a loss function, usually expressed as sum of square errors:

$$J(\mathbf{x}; W, \mathbf{b}) = \frac{||\mathbf{y}^{out}(\mathbf{x}) - \mathbf{y}^{true}||^2}{2N}, \qquad (1)$$

where W is the collection of all the weights $\{w^l_{jk}\}_{jkl}$, $j = 1, ..., N^l$, $k = 1, ..., N^{l-1}$, $l = 1, ..., L$ and $\mathbf{b}$ the vector of the biases $\{b^l_j\}_{jl}$, $j = 1, ..., N^l$, $l = 1, ..., L$, and $|| \cdot ||$ the Euclidean norm. The loss function of the output layer is given by (1), while in the hidden layers it is modified according to the backpropagation algorithm [8, 12].

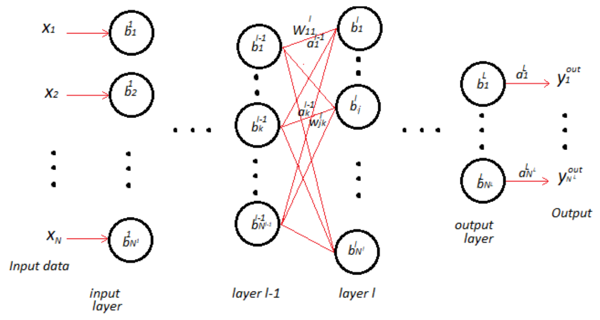

Figure 2: Scheme of a backpropagation network.

The performance of an ANN depends on the design of its architecture, based on the tuning of the so-called hyperparameters, that may include the number of layers and neurons, the activation function, etc. A popular choice of activation function is the sigmoid, i.e.

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{2}\left(1 + \tanh\left(\frac{z}{2}\right)\right).$$

Cybenko [2] proved that feedforward dense neural networks with sigmoidal activation functions are universal approximators for continuous functions (see [5] for details).
Extensions of this result to the estimation of measurable functions and functions belonging to Sobolev spaces and their derivatives are possible.

2.1. ANNs for the solution of PDEs

ANNs-based approaches to the numerical solution of PDE were proposed in [15].
Consider a generic boundary value problem with Dirichlet boundary conditions expressed in general form as:

$$\begin{cases} Lu(x) = f(x) & \text{in } \Omega, \\ u = g_D & \text{on } \partial\Omega, \end{cases} \qquad (2)$$

where L is a differential operator to be specified. A basic approach [3] consists in determining the approximate solution u_h as the output of an ANN taking as input suitable coordinate points in the domain and the corresponding evaluation of the PDE data, trained for learning the parameters $p = \{W, \mathbf{b}\}$ by minimizing the following loss function:

$$J(\mathbf{x}, p) = \sum_{i=1}^{n} [(L\Psi_t(x_i, \mathbf{p}) - f(x_i))^2 + \\ + (\Psi_t(x,p) - g_D(x_i))^2],$$

involving a penalty term for the enforcement of the boundary constraint.
The proposal of Lagaris et al. [7] is instead based on the following trial solution:

$$\Psi_t(x,p) = \hat{\Psi}(x) + F(x)N(x,p), \qquad (3)$$

where $\hat{\Psi}$ is a smooth extension of the boundary condition g_D, $F\big|_{\partial\Omega} = 0$ and $N(\cdot, p)$ is an ANN giving as output the approximate solution of the problem (2), trained without taking into account the Dirichlet condition, by minimizing the following loss function:

$$J(\mathbf{x}, p) = \sum_{i=1}^{N} [\nabla\Psi_t(x_i, p) - f(x_i)]^2. \qquad (4)$$

The approximate solution $\Psi_t(\hat{\mathbf{x}}, p^\star)$ of the PDE at any point $\hat{\mathbf{x}} \in \Omega$ is finally given by the trial solution with parameters $p^\star = \arg\min_p J(\mathbf{x}, p)$.
Other approaches based on the traditional Galerkin-based numerical methods rely on Finite Element Neural Networks (FEMNNs) [10], whose structure is built according to the mesh definition. Some authors have also applied deep learning to the discovery of the PDE expression based on a set of scattered data [9].
Finally, extensions to time-dependent problems were also proposed [13], in which the time is considered as an additional variable and the ANN minimizes the following loss function:

$$J(\mathbf{x}, t, p) = J_{GE}(\mathbf{x}, t, p) + J_{IC}(\mathbf{x}, p) + J_{BC}(\mathbf{x}, t, p),$$

where J_{GE} is given by equation (4), while J_{IC} and J_{BC} are penalization terms corresponding to the imposition of the initial and boundary condition respectively.

3. The network structure

My thesis is based on the work of Xu et al. [14], who propose to replace the deterministic field Ψ_t in (3) with another independent ANN that approximately satisfies the boundary condition: $A(x,y;w_1)\big|_{\partial\Omega} \approx g_D \ \forall (x,y) \in \partial\Omega$. Then, the expression of the approximate solution becomes:

$$u_h(x,y;w_1,w_2) = A(x,y;w_1) + \\ + B(x,y)N(x,y;w_2), \qquad (5)$$

where B is a smooth function whose restriction to the boundary is zero and N is a network approximating the PDE solution, whose parameters w_2 are learned by the minimization of the following loss function:

$$J_p(\mathbf{x}, \mathbf{y}; w_2) = \\ = \sum_{i=1}^{n_p} [f(x_i, y_i) - LN(x_i, y_i; w_2)]^2, \qquad (6)$$

where LN is the numerical evaluation of the differential functional of the problem (2) at the output of the PDE network N.
The boundary network $A(x,y;w_1)$ is independently trained on points belonging to $\partial\Omega$ only, updating its parameters w_1 by minimizing the boundary loss function:

$$J_b(\hat{\mathbf{x}}, \hat{\mathbf{y}}; w_1) = \\ = \sum_{i=1}^{n_b} [(g_D(\hat{x}_i, \hat{y}_i) - A(\hat{x}_i, \hat{y}_i; w_1)]^2. \qquad (7)$$

The coupled structure is trained in M iterations of a loop that consists in the minimization of (6) corresponding to one randomly sampled set of coordinate points $\{(x_n, y_n)\}_{n=1}^{n_p} \subset \Omega$ (training of the PDE network N) and the optimization of (7) corresponding to random samples $\{(\hat{x}_n, \hat{y}_n)\}_{n=1}^{n_b} \subset \partial\Omega$ (training of boundary network A).
After the training phase, the optimal networks parameters w_1^* and w_2^* are determined and the approximate solution of the PDE at any point of the domain can be computed as follows:

$$\tilde{u}_h(\tilde{x}, \tilde{y}) = u_h(\tilde{x}, \tilde{y}; w_1^*, w_2^*),$$

where u_h is given in (5).
For details about the coupling structure, see Figure 3.
If the problem has the following structure:

$$\begin{cases} u_t - Lu = f & \text{in } \Omega \times (0,T], \\ u = g_D & \text{on } \partial\Omega \times (0,T], \\ u(\mathbf{x}, 0) = g(\mathbf{x}) & \text{in } \Omega, \end{cases} \qquad (8)$$

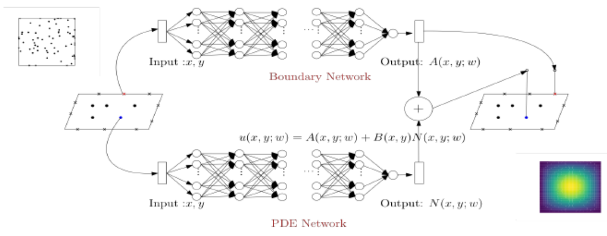

Figure 3: Coupled neural network structure. (Taken from [14])

I have proposed an extension of the proposed approach. The time variable t is treated as an additional data variable and impose the initial condition as a penalization term in the loss function of the PDE network.
The trial solution is modified in order to take into account also the variable t:

$$u_h(x, y, t; w_1, w_2) = A(x, y, t; w_1) + \\ + B(x, y)N(x, y, t; w_2),$$

where B is the same functional introduced in (3) and A and N are ANNs whose parameters w_1 and w_2 are learned by minimizing the following loss functions, respectively:

$$J_b(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{t}}; w_1) = \\ = \sum_{i=1}^{n_b} [(g_D(\hat{x}_i, \hat{y}_i.\hat{t}_i) - A(\hat{x}_i, \hat{y}_i, \hat{t}_i; w_1)]^2$$

and

$$J_p(\mathbf{x}, \mathbf{y}, \mathbf{t}; w_1) = \\ = \sum_{i=1}^{n_b} [(g(x_i, y_i) - N(x_i, y_i, t_i; w_1))^2 + \\ + (f(x_i, y_i, t_i) + \\ - N_t(x_i, y_i, t_i; w_2) + LN(x_i, y_i, t_i; w_2))^2].$$

The only difference in the training algorithm is that the samples are now $(d+1)$-dimensional, i.e. $\{(x_n, y_n, t_n)\}_{n=1}^{n_p} \subset \Omega \times [0, T]$ for the PDE network N and $\{(\hat{x}_n, \hat{y}_n, \hat{t}_n)\}_{n=1}^{n_b} \subset \partial\Omega \times [0, T]$ for the boundary network A.
The networks architecture, i.e. the choice of the number of layers and neurons, is set up by trial and error. I have tested the method performance with networks made of 1, 2 and 3 layers composed of 128, 256 and 512 neurons each. As activation function I have chosen the hyperbolic tangent $\sigma(\mathbf{x}) = \tanh(\mathbf{x})$ for all the hidden layers and the linear function $\sigma(\mathbf{x}) = \mathbf{x}$ for the output layer.
Finally, the adopted optimization algorithm is a variation of the gradient descent, called Adam optimizer, based on the adaptation of the learning rate at each iteration, in order to speed up the convergence and reduce the probability of settling into a local minimum [11].
As a measure of the approximation error, I have tested the method on a 20×20 spatial grid over Ω and on a $20 \times 20 \times 20$ grid over $\Omega \times [0, T]$, evaluating its l^2 norm, defined in the spatial and time-dependent cases, respectively, as follows:

$$err_2 = \sqrt{\frac{1}{m} \sum_{i=1}^{m} |u_h(x_i, y_i; w_1, w_2) - u(x_i, y_i)|^2}, \tag{9}$$

and

$$err_{2,t} = \\ = \sqrt{\frac{1}{m} \sum_{i=1}^{m} |u_h(x_i, y_i, t_i; w_1, w_2) - u(x_i, y_i, t_i)|^2}. \tag{10}$$

4. Numerical results

The architecture of the networks is chosen to be composed of 3 hidden layers made of 256 neurons each, and the initialization of the learning rate of the Adam optimizer is fixed to $\eta = 0.001$.

4.1. Poisson problem

The first test case (TC.E1) presents a smooth analytical solution $u(x, y) = \sin(\pi x)\sin(\pi y)$, that solves the problem

$$\begin{cases} \Delta u = f & \forall (x, y) \in \ \Omega = (0, 1)^2, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

with f derived accordingly.
The second test case (TC.E2) presents instead a steep peak in the centre of the domain and its exact solution has the following expression:
$u(x, y) = e^{-1000(x-0.5)^2 - 1000(y-0.5)^2}$ in $(0, 1)^2$.
Figure 4 shows the plot of the error defined as in (9) as a function of the training iteration counts. It decreases and reaches 10^{-3} in almost 300 iterations and then starts oscillating around values of order 10^{-4}.

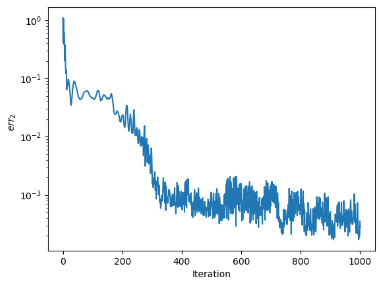

Figure 4: TC.E1: error as a function of the iteration counts.

In Figure 5 we can see that no convergence is attained in 1000 iterations for the test case (TC.E2): after an initial decrease, the error keeps oscillating around 10^{-1}. This is due to the high-gradient of the exact solution near the peak, where it suddenly becomes very large. Indeed, the plot on the right shows that the approximate curve even presents an opposite curvature to the expected one.

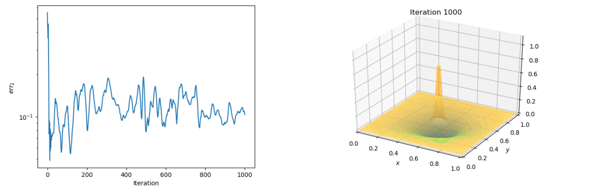

Figure 5: TC.E2: error as a function of the iteration counts (left) and computed (green) and exact (orange) solutions after 1000 training iterations (right).

The exact solutions of test cases (TC.E1) and (TC.E2) are extremely regular, in $C^\infty(\Omega)$. Therefore, the approximation error in the L^2 norm of the Finite Element method based on a mesh of granularity h is controlled by h^2 and reaches the same order as the convergence value of the ANN-based method in both cases in the same computational time and with meshes made of almost 10^3 nodes.

4.2. Heat equation

The last problem analyzed (TC.P1) is a special case of system (8), where the differential operator L is specified as the Laplacian and the exact solution is chosen as $u(x, y, t) = \sin(\pi x) \sin(\pi y)e^{-t}$ on the domain $\Omega = (0, 1)^2$. The initial and boundary data are set accordingly.

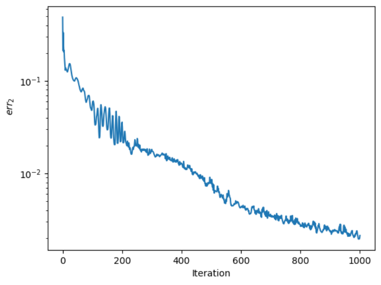

Figure 6: TC.P1: error as a function of the iteration counts.

In Figure 6 we can notice a lower learning rate of the networks coupling, maybe due to the additional penalization term in the PDE loss concerning the initial condition. Within 1000 iterations the error (10) reaches at most order 10^{-2}, even if it shows a continuously decreasing trend. Moreover, the plot presents very small oscillations, proving that the hyperparameters choice is valid also for the time-dependent case.

5. Conclusions

In the thesis we have extensively studied the performance of the ANN-based method of Xu et al. [14] for the approximate solution of PDEs. I have also extended to parabolic and hyperbolic problems the approach of Xu and tested it on problems having exact solutions with different levels of regularity.
The error tends to quickly settle around the convergence value, that is no lower than 10^{-3} in all the considered cases. Moreover, as we can deduce from the analysis of the problem with solution with a peak (Section 4.1), the method is also not able to detect particular features of the solution in small areas of the domain.
This is however a good black-box approach for the approximate solution of PDEs, even if it cannot substitute the traditional methods.
Possible future developments may follow two different paths: either trying to optimize the networks structures by making use of some autotuning techniques, or moving the focus on the improvement of Galerkin-type scheme trying to calibrate the values of some related parameters via physics-informed PDEs.

6. Acknowledgements

I would like to express my gratitude to my advisor, Professor Paola Francesca Antonietti, for her precious support and directions during the research for my thesis and for showing me many interesting unknown aspects and applications of this subject.

References

[1] Susanne C. Brenner and L. Ridgway Scott. *The mathematical theory of finite element methods*, volume 15 of *Texts in Applied Mathematics*. Springer, New York, third edition, 2008.

[2] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[3] Gamini M. W. M. Dissanayake and Nhan Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[4] A. Ronald Gallant and Halbert White. On learning the derivatives of an unknown mapping with multilayer feedforward networks. *Neural Networks*, 5(1):129–138, 1992.

[5] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[6] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5):551–560, 1990.

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

[8] Yann LeCun. A theoretical framework for back-propagation. In David Touretzky, Geoffrey Hinton, and Terrence Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, volume 1, pages 21–28. Morgan Kaufmann, 1988.

[9] 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. *Journal of Computational Physics*, 378:686–707, 2019.

[10] Pradeep Ramuhalli, Lalita Udpa, and Satish S. Udpa. Finite-element neural networks for solving differential equations. *IEEE Transactions on Neural Networks*, 16(6):1381–1392, 2005.

[11] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv e-print*, 1609.04747, 2017.

[12] David E. Rumelhart, Geoffrey E. Hinton, and Rondald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[13] Justin Sirignano and Konstantinos Spiliopoulos. DGM: a deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

[14] Kailai Xu, Bella Shi, and Shuyi Yin. Deep learning for Partial Differential Equations (PDEs). 2018.

[15] Neha Yadav, Anupam Yadav, Manoj Kumar, et al. *An introduction to neural network methods for differential equations*. Springer, 2015.