One-Dimensional Wave Equation Simulation Using Recurrent Neural Networks

Maurício D. Silva[1], Lavínia M. Takarabe[1], Carolina Benetti[1], Anderson G. Santiago[1]

[1]*Federal University of ABC, Center for Engineering, Modeling and Applied Social Sciences*
Alameda da Universidade, s/n - Anchieta, São Bernardo do Campo - SP, Brazil
{mauricio.devincentis, lavinia.mitiko}@aluno.ufabc.edu.br, {c.benetti, gabriel.santiago}@ufabc.edu.br

Abstract. This paper presents an application of Recurrent Neural Networks to the one-dimensional Wave Equation. Nowadays, Neural Networks have been widely used due to the advances in computer hardware, since it allows that a great amount of data could be processed in parallel. Over the years, new and improved neural network architectures were developed, as for example the Recurrent Neural Network, mainly used in time series analysis. In this study, the 1-D wave equation solution is implemented using Finite Differences in Time Domain considering Neumann Boundary Conditions, and two architectures of Recurrent Neural Networks were explored: LSTM and GRU. The results were organized according to the hyper-parameters used to train and validate the networks, and they were evaluated quantitatively, using the mean squared error as loss function, and qualitatively, observing the response plots for dataset validation. It was possible to achieve predictions with mean squared errors of order 10^{-6} and a training time of 23 seconds per epoch using GPUs.

Keywords: Wave Equation, Recurrent Neural Networks, Finite Difference Method, LSTM, GRU

1 Introduction

In the last years, Neural Networks (NNs) algorithms have gained notoriety due to their high capacity in solving complex problems such as image and speech recognition and medical image processing. Furthermore, they had proved to be a powerful tool in the Numerical Calculus field: one of the first implementations of NNs aiming at this kind of problem was presented by Lee and Kang [1] for parameter optimization of the Finite Difference Method (FDM) used to solve Partial Differential Equations (PDEs) as in Lima [2].

Since then, new NNs architectures were proposed in order to evaluate differential equations, such as the Wave Equations in Hughes et al. [3] and the Schrodinger Electronic Equations in Hermann et al. [4]. As expected, such studies showed solid results, since the Universal Approximation Theorem states that any continuous function can be approximated by a NN with hidden layers and a finite number of neurons as proposed by Lima [2].

Recently, specific NN architectures were developed to address the solution of time series problems, such as the Recurrent Neural Network (RNN). The RNN is characterized by its high capability of sequential data processing, as discussed by Goodfellow et al. [5], and is suitable for modeling wave equation solutions, as demonstrated by Hughes et al. [3].

In its original format, simple RNNs have a problem characterized by the short-term memory, meaning that, at a certain point, the network will fail to predict new values if the model requires information from the earlier steps. Two possible solutions are given by the development of networks architectures such as Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) (Nguyen [6]). Both of these architectures present similar concepts and aim to solve the short-term memory problem by implementing an intern mechanism known as ”gates”, responsible to perform the sequence learning in what can be called a ”smart way”: understanding which data the network must keep or discard according to some given criterion as in Goodfellow et al. [5].

The main difference between GRU and LSTM is that the first possesses only two gates in its structure, the ”Update Gate” and the ”Reset Gate”, while the latter has these two gates and at less two additional gates: the ”Forget Gate” and ”Output Gate” as described by Nguyen [6]. It is important to highlight that these gates are also Neural Networks and have their own characteristics. Figure 1 presents a comparison between these two architectures.

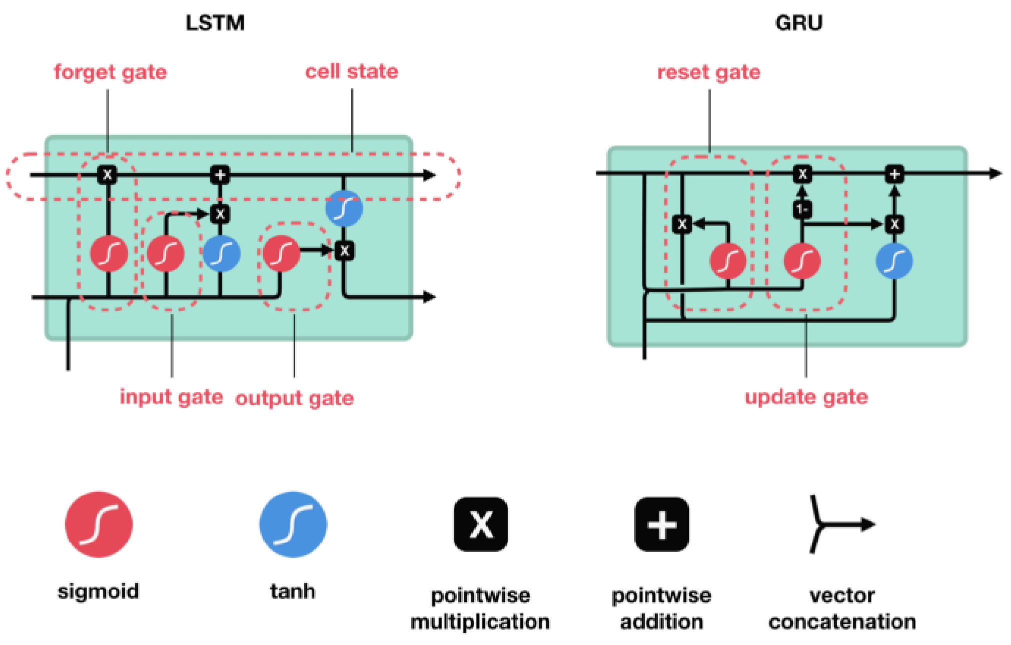

Figure 1. LSTM Architecture *versus* GRU Architecture[6].

Since these architectures are specialized in modeling time series problems, they have been used to solve differential equation systems along with numerical methods, such as the Finite Difference Method in Time Domain (FDM-TD) as mentioned by Yao and Jiang [7]. Thus, this paper presents a performance comparison between RNN-GRU and RNN-LSTM in predicting the response of the One-Dimensional Wave Equation solution for linear and homogeneous medium and Neumann boundary conditions. The dataset used in training and validation procedures was evaluated using FDM-TD and several hyperparameters were considered, such as the dimensionality of the output space, the number of learning units, the batch size, and the activation function.

2 Methodology

2.1 Wave Equation Solution and Dataset Preparation

The implementation of the FDM-TD and all RNN architectures were performed in Python 3.8 Anaconda distribution[1] along with NumPy[2], Tensorflow 2.1[3], and Matplotlib[4] packages. The One-Dimensional Wave Equation, its boundary, and initial conditions are given by Equations 1 to 5:

$$\frac{\partial^2 u(x,t)}{\partial t} = c^2 \frac{\partial^2 u(x,t)}{\partial x^2} \tag{1}$$

$$u(x=0,t) = \sin(\omega t) \tag{2}$$

$$\frac{\partial u}{\partial x}(x=L,t) = 0 \tag{3}$$

$$u(x,0) = 0 \tag{4}$$

[1]https://www.anaconda.com/
[2]https://www.numpy.org/
[3]https://www.tensorflow.org/
[4]https://www.matplotlib.org/

$$\frac{\partial u(x,0)}{\partial t} = 0 \tag{5}$$

Since the main concern of this paper is to compare LSTM and GRU architectures in predicting the wave equation solution, a standardized set of wave parameters were considered: $T = 1(s)$ and $\lambda = 1(m)$ with $c = 1(m/s)$; and for the FDM-TD model $\Delta x = \lambda/20(m)$, $\Delta t = T/20(s)$, $L = 2\lambda(m)$ within a time window of $2T(s)$.

2.2 Test and Validation Procedures and Neural Network Setup

The FDM-TD solution resulted in a matrix $\boldsymbol{U} \in \Re^{Nx \times NT}$, with Nx and NT the number of samples in x and t-axis respectively. The matrix was converted into an array $\bar{\boldsymbol{U}} \in \Re^{NxNT \times 1}$, as presented in Fig. 2.

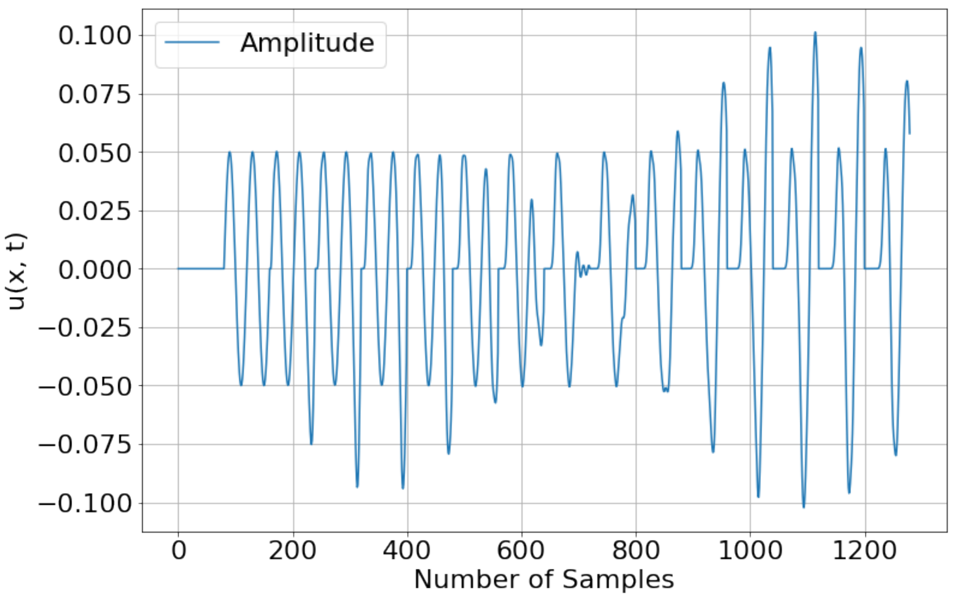

Figure 2. Wave Equation solution used as dataset for training and validation.

For training and validation purposes, a 95:5 ratio was used, i.e., using 95% of the available data for training (1292 samples) and 5% for validation (68 samples). In order to improve the training and validation results, a pre-processing stage was considered, consisting of a min-max normalization. Once the training and validation procedures were done, the dataset denormalization was applied in order to provide the actual predicted values.

The loss function considered in this paper was the ”mean squared error”, and the Adam optimizer developed by Kingma and Ba [8] with adaptive learning rate was adopted. For each architecture, the training procedure had a series of adjustable parameters as presented in Table 1 along with their respective range values. It is important to note that the same set of hyper-parameters were considered for both architectures, allowing comparison of their performances.

Table 1. Hyper-parameters set and values range

Hyper-parameter	Values Range
Nt	[2, 16]
Nu	[10^2, 10^3]
Length	[10, 50]
Batch	[1,50]

The activation functions considered are shown in Fig. 3.

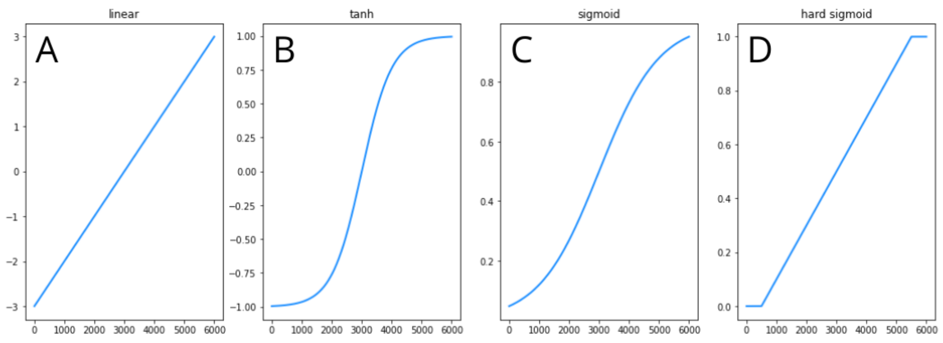

Figure 3. Activation functions considered: A) $linear$; B) $tanh$; C) $sigmoid$; D) $hard\_sigmoid$

For hyper-parameters optimization, an initial random exploration was considered taking into account the loss function evaluated for the training and validation datasets. Once a sub-optimal range for each hyper-parameter was defined, a grid search was applied, in which the hyper-parameters varied progressively and equally spaced. It is important to note that the number of epochs used in the experiments was not considered a hyper-parameter, since it was possible to monitor the validation error using the `early_stopping`[5] method.

3 Results

This section presents the results considering the $tanh$ and $hard\_sigmoid$ activation functions, since they presented the lowest loss function values for the training and validation dataset. Regarding the evaluation time for the whole training and validation procedures, both architectures were comparable, having an average time of $23(s/epoch)$. The optimal set of hyper-parameters for both architectures was observed to be the same and it is presented in Table 2.

Table 2. Optimal hyper-parameters set

Parameters	$tanh$	$hard\_sigmoid$
Nt	8	8
Nu	100	100
Batch	1	1
Length	10	10

3.1 GRU

Table 3 shows the loss function values for test and validation datasets and Fig. 4 presents the qualitative results.

Table 3. GRU: Minimum loss function errors observed

Dataset evaluated	$tanh$	$hard\_sigmoid$
Train	183.0×10^{-6}	402.0×10^{-6}
Validation	29.0×10^{-6}	188.0×10^{-6}

[5]Available at: https://keras.io/api/callbacks/early_stopping/

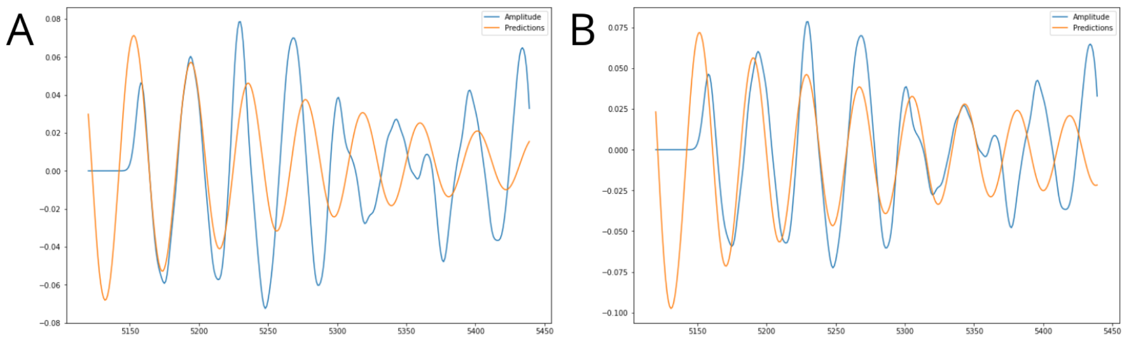

Figure 4. Activation function and GRU performance: A) $hard\_sigmoid$; B) $tanh$.

Figure 5 presents the variation of the loss function value for training and validation datasets. Each plot was evaluated by varying one of the parameters while keeping all others fixed with values presented in Table 2.

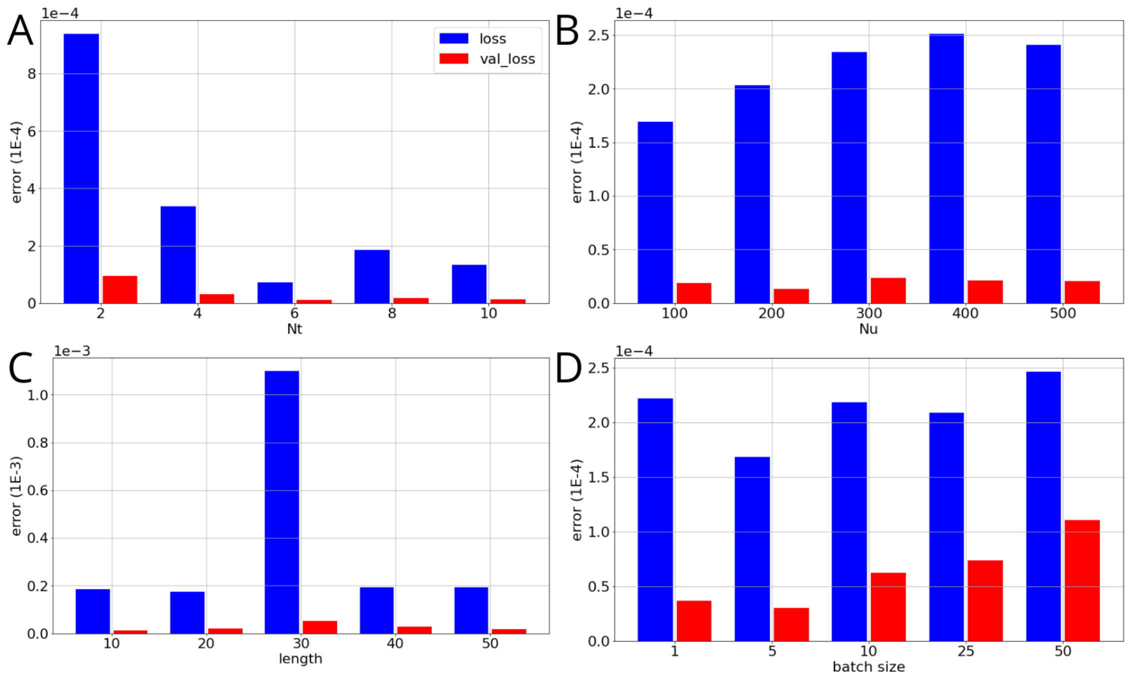

Figure 5. Error profile for GRU considering as variable: A) Nt; B) Nu; C) Length; D) Batch Size;

3.2 LSTM

Table 4 shows the loss function values for test and validation datasets and Fig. 6 presents the qualitative results.

Table 4. LSTM: Minimum loss function errors observed

Dataset evaluated	$tanh$	$hard\_sigmoide$
Train	182.0×10^{-6}	295.0×10^{-6}
Validation	31.0×10^{-6}	638.0×10^{-6}

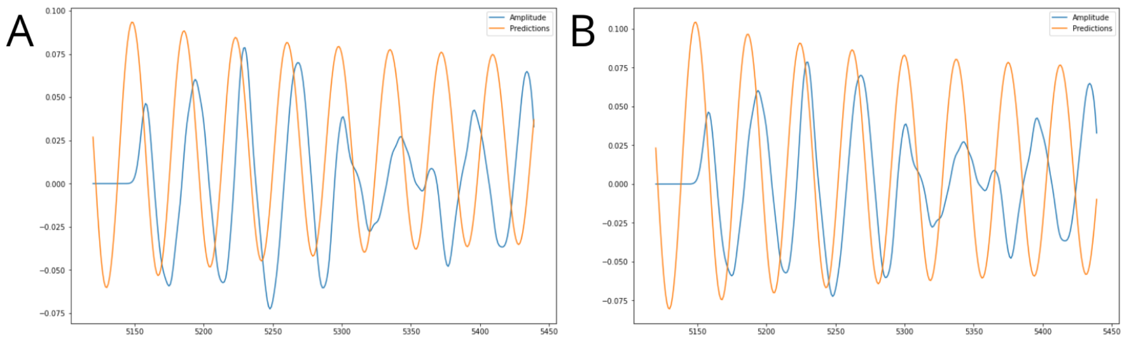

Figure 6. Activation function and LSTM performance: A) $hard\_sigmoid$; B) $tanh$.

Figure 7 presents the variation of the loss function value for training and validation datasets. Each plot was evaluated by varying one of the parameters while keeping all others fixed with values presented in Table 4.

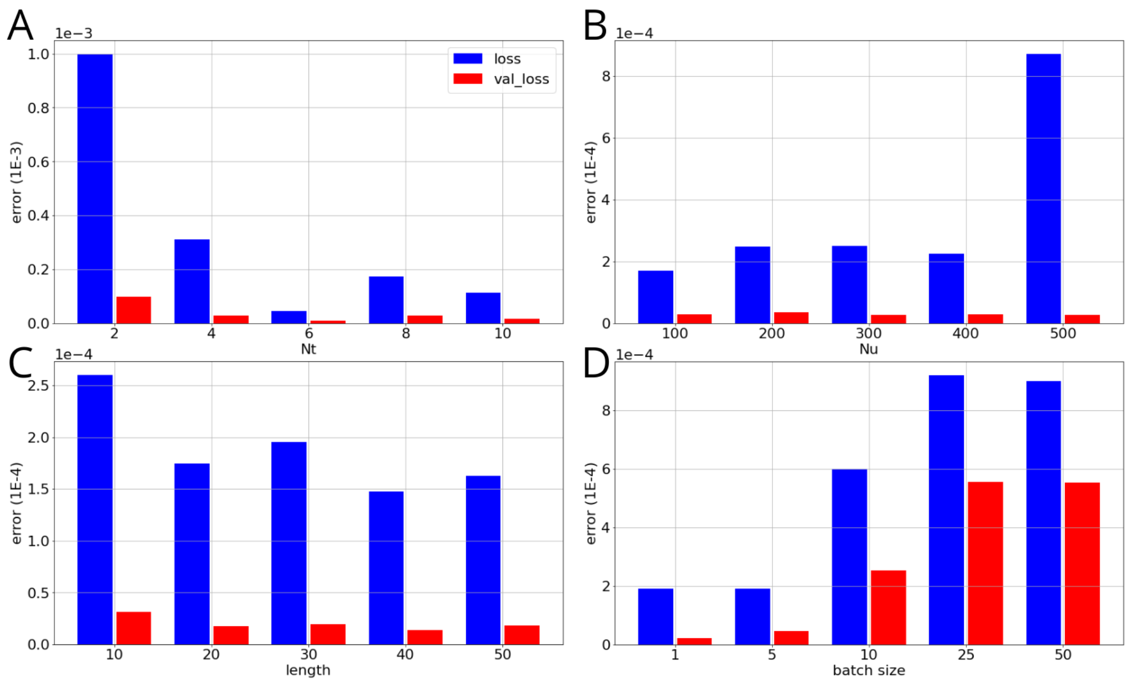

Figure 7. Error profile for GRU considering as variable: A) Nt; B) Nu; C) Length; D) Batch Size

4 Discussion

As presented in the previous section, the GRU and LSTM presented similar performances with the same hyper-parameters set. From a qualitative point of view, i.e., observing Figs. 5 and 7, GRU with $tanh$ activation functions was able to reproduce the frequency of the validation dataset and even match its amplitude in several points. The same conclusion can be said when taking into account the loss function value evaluated for the validation dataset: GRU and $tanh$ presented the lowest value considering all four experiments presented.

It can be observed that, despite the loss function values presented in Table 4, the LSTM architecture along $tanh$ and $hard\_sigmoid$ presented an almost constant amplitude, while the validation dataset presented a decaying; it also presented a phase shifting as can be observed in Figs. 7.

5 Conclusion

This paper presented a study of the performance of LSTM and GRU Recurrent Neural Networks in predicting values for the One Dimensional Wave Equation with Neumann boundary condition with a sine wave input. The results presented suggest that both the LSTM and GRU architectures can be used to simulate the wave equations, although a more suitable loss function must be used in order to accurately describe and penalize the amplitude and phase shifting errors. Also, more studies regarding the impact of the variation of the hyper-parameters values over the amplitude, frequency and phase can be designed in order to explain how the networks lean the considered time series.

Acknowledgements. The authors would like to acknowledge the Federal University of ABC.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] H. Lee and I. S. Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, vol. 91, n. 1, pp. 110–131, 1990.
[2] L. Lima. *Numerical Solution of PDE's Using Deep Learning*. PhD thesis, 2019.
[3] T. W. Hughes, I. A. Williamson, M. Minkov, and S. Fan. Wave physics as an analog recurrent neural network. *Science advances*, vol. 5, n. 12, pp. eaay6946, 2019.
[4] J. Hermann, Z. Schätzle, and F. Noé. Deep-neural-network solution of the electronic schrödinger equation. *Nature Chemistry*, vol. 12, n. 10, pp. 891–897, 2020.
[5] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.
[6] M. Nguyen. Illustrated guide to lstm's and gru's: A step by step explanation. "https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21". [Online; accessed 19-July-2021], 2018.
[7] H. M. Yao and L. J. Jiang. Machine learning based neural network solving methods for the fdtd method. In *2018 IEEE International Symposium on Antennas and Propagation & USNC/URSI National Radio Science Meeting*, pp. 2321–2322. IEEE, 2018.
[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.