NEURAL NETWORK SOLVER FOR THE ONE DIMENSIONAL SHALLOW WATER EQUATIONS

By

Youssef Zaazou

Thesis Project
Submitted in Partial Fulfillment of the
Requirements for the Degree of

Bachelor of Science
Department of Mathematics and Statistics
Department of Physics and Physical Oceanography

April 2021

Dr. Alex Bihlo, First Reader

Dr. Candemir Cigsar, Second Reader

Memorial University of Newfoundland

ABSTRACT

Neural networks (NN), particularly deep NNs, have emerged in the last decade or so as an invaluable and powerful computational tool that enables us to make sense of the incredible amounts of data being generated today. Their applications are nearly endless, from computer vision to bank fraud detection, it seems there is very little a NN cannot be trained to do. A newly emerging field is that of using NNs to solve differential equations encountered in computational science and engineering. One such set of equations are the shallow water equations (SWEs).

The SWEs are a simple, yet effective, model consisting of a set of hyperbolic differential equations with applications in climatology and tsunami modelling. Traditionally, models based on the SWEs have been solved using numerical methods including finite-difference methods, finite-volume methods and spectral element methods. These methods can be time consuming and computationally expensive, not desirable qualities especially when attempting to model tsunamis in real time. Therefore, we will investigate the use of NNs for solving the SWEs in one dimension.

Our first order of business is determining the architecture of the NN best suited to solve the SWEs. Using traditional numerical methods, specifically, the Lax-Wendroff scheme, we generate datasets consisting of solutions for the SWEs from which our NNs will extract the physical relations embedded in the data. Our dataset encompasses a variety of initial conditions which allows the network to generalize and avoids overfitting. Using this data, we train different architectures of networks including feedforward, recurrent and generative adversarial networks and investigate the predictions of each.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

We thank Dr. James Jackaman for his help and support regarding the numerical solution of the shallow-water equations, as well as useful discussions, numerous insights and recommendations throughout this project.

CHAPTER 1

INTRODUCTION

The shallow water equations (SWE) exhibit some essential characteristics of fluid flow where the horizontal length scale is significantly greater than the vertical length scale. They have been traditionally used in tsunami modelling, see e.g. [1], river flooding, see e.g. [2], and atmospheric modelling, see e.g. [3]. In all of these areas of application, numerical schemes have been used historically, and while computation times have been greatly reduced due to improvements in computing resources, these schemes remain computationally expensive to run. Quite recently machine learning has started to be touted as a possible replacement for numerical schemes, see e.g. [4]. Machine learning algorithms can offer considerable advantages over their numerical counterparts; once a deep neural network is trained it can produce predictions at a fraction of the computational cost of traditional differential equation based numerical schemes.

It is well established that neural networks (NN) can function as universal approximators, see [5], that can implicitly detect complex and nonlinear patterns in a given data set. Some NN architectures are more suited for certain applications and types of data. For instance, convolutional neural networks are particularly suited for analyzing visual imagery, see [6], such as image classification and segmentation due to their shift invariance, see [7]. Time series prediction however would favor the employment of a recurrent neural network due to their memory state which enables them to exhibit temporal dynamic behavior, see [8].

In this work, we investigate the suitability of several NN architectures in solving the one dimensional SWEs. They include traditional feedforward networks, recurrent NNs and generative adversarial networks (GAN). We are looking for the architecture best suited to be a universal approximation function that will learn the complex nonlinear relationship between input and output data.

CHAPTER 2

PROBLEM AND SETUP

2.1 Neural Networks

Before discussing neural networks, we will explain the function of a single neuron and how it works. A neuron is a mathematical operation that outputs a certain activation given an input. First, an input vector X is received by the neuron where it is then dotted with a weight vector W that is stored in the neuron. W's components assign a weight to each input allowing the neuron to select which signals pass through to the output and how much each particular signal is detectable in the output. The output of this dot product is then passed into an activation function, such as the $tanh$ or $sigmoid$ function, the result of which becomes the output of the neuron.

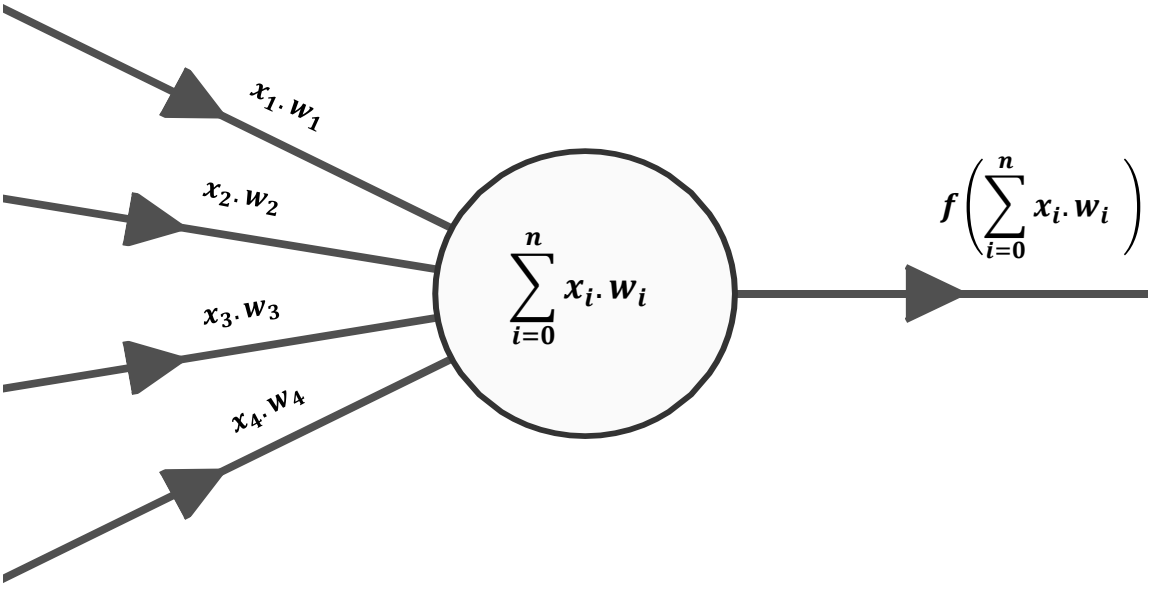


Figure 2.1: Overview of a single neuron.

If we take a number of these neurons and arrange them in a stack, this stack is referred to as a

layer. Take multiple layers and arrange them in a sequential manner, and we get a neural network that is referred to as a feedforward neural network. A feedforward neural network propagates its input by passing it along one layer to another in a sequential manner. To clarify, in a feedforward architecture the neurons in each layer are not connected to each other rather each neuron in a given layer receives input from each neuron in the previous layer and passes it's output to each neuron in the following layer. To put it in mathematical terms, a feedforward network is a nonlinear transformation that maps an input vector to an output vector. The transformation coefficients are the weights of the neurons which are referred to as the the learned parameters since it is these parameters that are acquired through the learning process.

We are training a network using supervised learning, where a network is fed data in input-output pairs so that it may learn the mapping from input to output. We will use the feedforward architecture to give an overview of the training process, but the basic principles used for this type of network apply to more complex network architectures that we will go over in subsequent chapters.

First, the input vector is propagated through the network and an output vector is obtained. This output vector along with the desired output (label vector) are then fed into a loss function, such as the mean squared error function (MSE), which calculates the distance between the network's output vector and the label vector. After that, the actual learning process begins with the backpropagation algorithm.

Backpropagation, see[8], is, in essence, a minimization algorithm where we are looking for the weights in the network that would minimize the loss function. This is done by computing the gradient of the loss function with respect to the weights and using an optimization algorithm to update the weights appropriately. Examples of optimizers include gradient descent where repeated steps are taken in the negative direction of the gradient to locate a local minimum of the loss function. In practice, more sophisticated optimizers such as stochastic gradient descent and the Adam optimizer are used due to their efficiency in high dimensional spaces compared to gradient descent. Once the optimal weights are obtained the network is updated with these weights and the process is, for a single input-output pair, complete.

This process is repeated with multiple data pairs until our network has generalized to the class of all possible input data. That means that our network should perform, on average, just as well with unseen data as with data that was used to train it. If the performance of the network is much better with training data than with testing data, then we have overfitting which indicates that the network is too powerful (has to many neurons/ parameters) for the given data set and is therefore memorizing the data by heart rather than extracting the relations embedded in the data. The concept of overfitting NNs is similar to that of overfitting in polynomial regression where a polynomial of too high a degree is selected to fit the data and so fits the data perfectly but is not an adequate model.

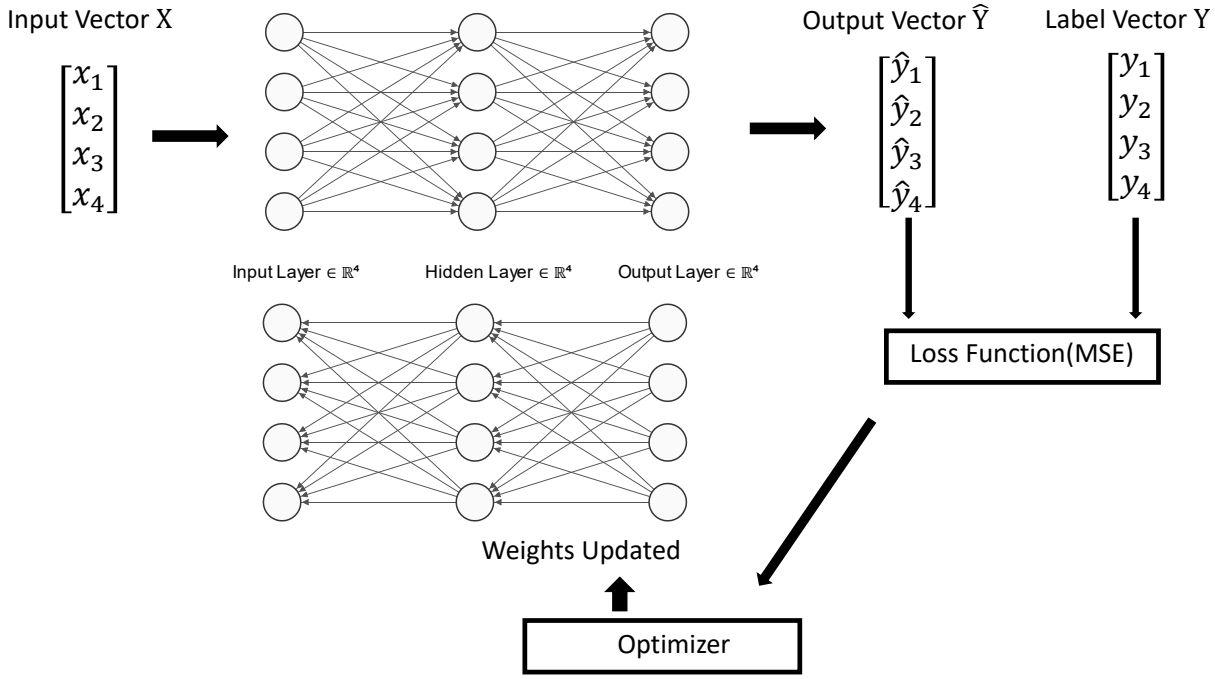


Figure 2.2: Flow chart outlining training process for a single input-output pair.

2.2 The Shallow Water Equations

The one dimensional shallow water equations:

$$\frac{\partial H}{\partial t} + \frac{\partial (HU)}{\partial x} = 0, \tag{2.1}$$

$$\frac{\partial (HU)}{\partial t} + \frac{\partial}{\partial x}(U^2 H + \frac{1}{2} H^2 g) = 0, \tag{2.2}$$

are a set of hyperbolic partial differential equations that are a reduction of the two dimensional version:

$$\frac{\partial H}{\partial t} + \frac{\partial (HU)}{\partial x} + \frac{\partial (HV)}{\partial y} = 0, \tag{2.3}$$

$$\frac{\partial (HU)}{\partial t} + \frac{\partial}{\partial x}(HU^2 + \frac{1}{2} H^2 g) + \frac{\partial}{\partial y}(HUV) = 0, \tag{2.4}$$

$$\frac{\partial (HV)}{\partial t} + \frac{\partial}{\partial x}(HUV) = 0 + \frac{\partial}{\partial y}(HV^2 + \frac{1}{2} H^2 g). \tag{2.5}$$

U and V are the x and y components of the mass velocity respectivly, H is the surface height and g is gravitational acceleration. The two dimensional SWEs are themselves derived from the Navier-Stokes equations.

Since NNs act as universal function approximators, we can use a NN to approximate solutions for the SWEs. This is done by feeding the NN a solution for the SWEs at time t and training it to map this input to time $t + i\Delta t$, where i denotes the i'th time step. To do this we will first need to obtain reliable training data arranged in pairs of input-output vectors that contain the physical laws we wish our network to learn.

2.3 Data Generation

We use traditional numerical methods to generate our training and testing data. For this we employ the Lax-Friedrichs method, see [9], a finite difference scheme that is explicit and second order

accurate in time and space and stable given that the Courant–Friedrichs–Lewy condition is satisfied[10]. We discretize the domain and use periodic boundary conditions to obtain our solutions. We must obtain a data set that is representative of the collective set of physical laws implied by the SWEs, this is so that our network can generalize well and avoid underfitting. To achieve that, we run the scheme for a large variety of initial conditions (IC) by employing a wave generating function that populates H at $t = 0$ with a random collection of Gaussian bumps each time it is called. U, is always zero at $t = 0$ which ensures a stable calculation that does not blow up or contain any spurious elements.

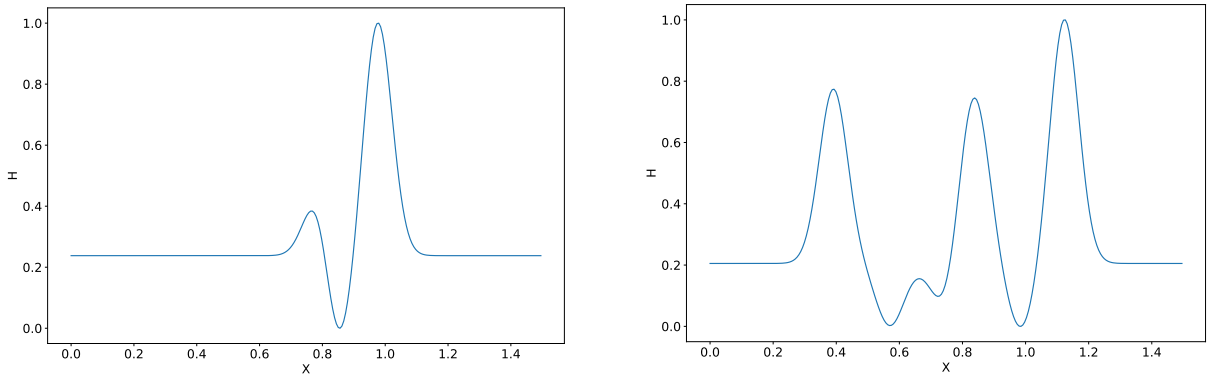


Figure 2.3: Examples of Gaussian bumps generated as ICs

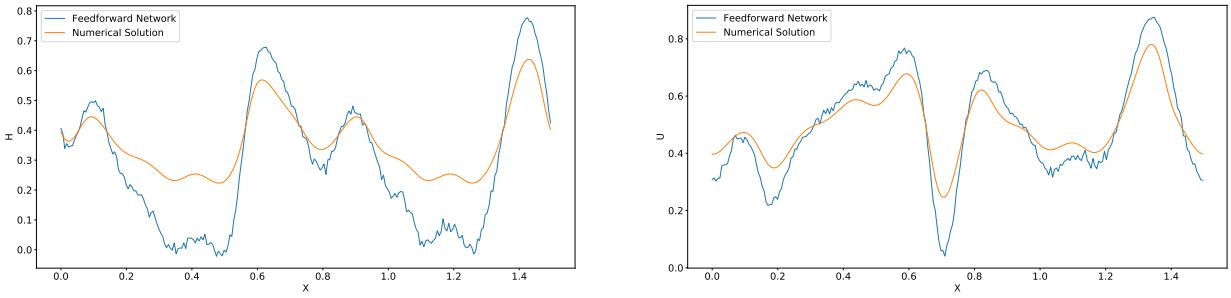
We combine the data obtained by running our numerical method for several ICs into one dataset that is then split into a training set and testing set. Splitting the data set like this is standard practice and ensures that we have minimal overfitting. For a thorough explanation of overfiting and underfitting and how they relate to the Bias and Variance dilemma see [11]. We then normalize the data at each time step to ensure that we do not encounter any issues with the stability of our NN during training.

In the following chapters we go over the NN architectures that we tested beginning with the feedforward architecture.

CHAPTER 3

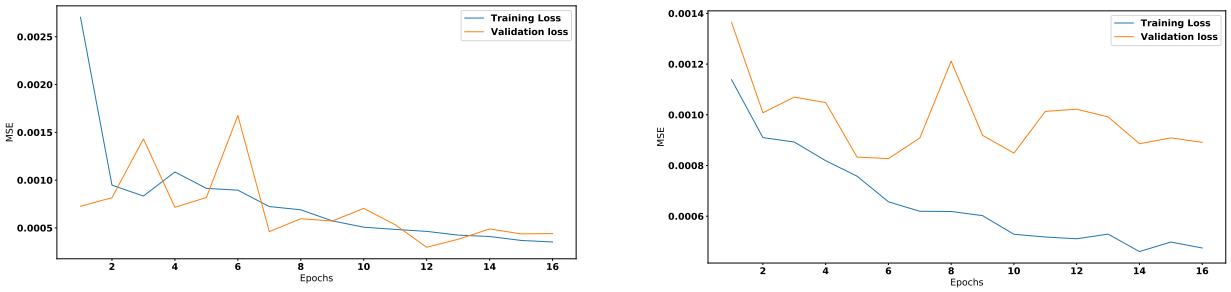
FEEDFORWARD NETWORK

The feedforward architecture is the most straight forward NN and as such is a great baseline model to which we can compare more sophisticated ones. We have found that hyperparameters best suited with this architecture are as follows: six densely connected layers each with as many nodes as there are spatial sampling points. We also have a dropout layer(s) with a dropout rate of 0.25 to curb overfitting.



(a) Height (b) Velocity

Figure 3.1: Predictions of Feedforward Network



(a) Height (b) Velocity

Figure 3.2: Training and Validation losses

This network clearly has a hard time learning the mapping from one time step to the next. The main features of the input wave are present in the network's output, but given the discrepancy in

amplitude between the network's output and the label and the the noise plaguing the output signal, the results of the network are unusable.

Also, plotting the losses as a function of epochs, we see that the behavior of the validation loss is erratic with sharp spikes throughout. Note how far the curve for the validation loss is from the training loss for the velocity indicating that the network has a hard time generalizing. This is all to be expected since feedforward networks are not very well suited to handle time series data.

CHAPTER 4

RESIDUAL NETWORK

An obvious candidate to look at is a recurrent neural network, due to their suitability with time series data, but before that we investigate a slight modification to the feedforward architecture. Residual networks, see [12], are particularly suited to our problem due to the skip connections which work as follows: two copies of the input vector, X, are made at the start of the network. Each copy goes through a different transformation: the first copy is propagated through a number of weight layers $\mathcal{F}$ as in the feedforward architecture while the second copy is simply multiplied by the identity function meaning x is left unchanged. The output of the weight layers $\mathcal{F}(X)$ is added to the output of identity mapping, the result of that addition is then passed through an activation function. This whole operation is a single residual block and we can stack multiple blocks where the output of one block is the input of the next. In our network we used three blocks each containing two weight layers for a total of six weight layers as in the feedforward network.

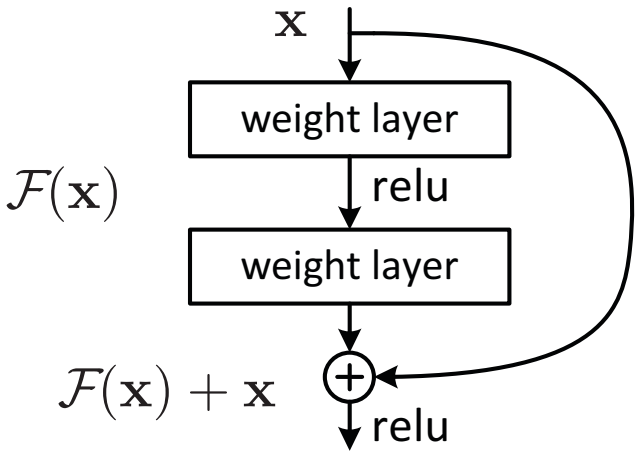
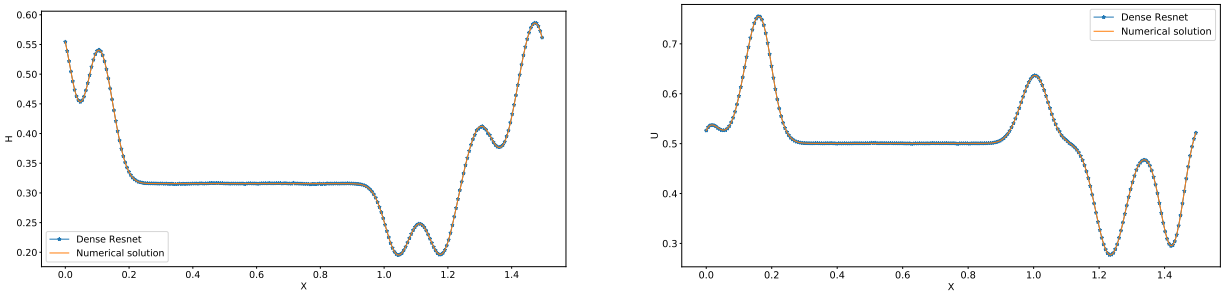


Figure 4.1: Schematic of skip connection.

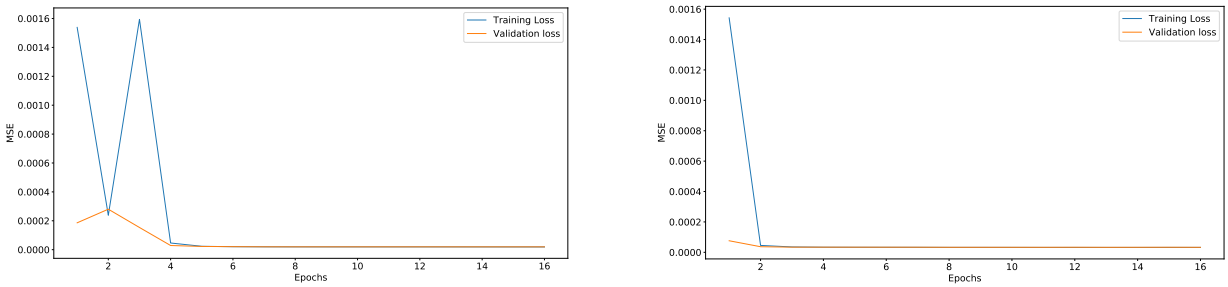
One of the main reasons the feedforward network was having a hard time learning was that it was required to construct the output from scratch which is computationally expensive and unreliable. Given the nature of the problem, the difference between the input vector and output vector is

not large at all; in fact, all that changes from one time step to another is that the waves themselves are translated ever so slightly along with some minor dissipation and dispersion. Our network would have a much easier time learning the changes in the data rather than generate the data itself and that is what these skip connections enable. They allow our network to employ, directly, the input when determining the output and thus can be trained more efficiently and produce more accurate results.



(a) Height (b) Velocity

Figure 4.2: Predictions of Residual Network



(a) Height (b) Velocity

Figure 4.3: Training and Validation Losses

We observe a significant improvement over the feedforward network with the output being free of any visible noise and much better convergence to the numerical solution. Taking a look at the training error we note that convergence was achieved much faster than with the feedforward network. These are both indications these skip connections, as predicted, are very well suited to our problem.

CHAPTER 5

LONG SHORT-TERM MEMORY

We can take advantage of the sequential nature of the data by using a recurrent neural network (RNN) to make our predictions. RNNs are specifically suited to time series data owing to their ability to preserve information across time steps. Figure 5.1, where A represents a weight layer, x_t the input vector and h_t the output vector at time t, provides an explanation of how an RNN preserves information. A has built within it a feedback loop that allows the state of A to be passed to the next time step. This internal state can then be thought of as memory. The result of unrolling this loop in time can be seen on the right hand side where each state of A at time t is passed on to the next time step.

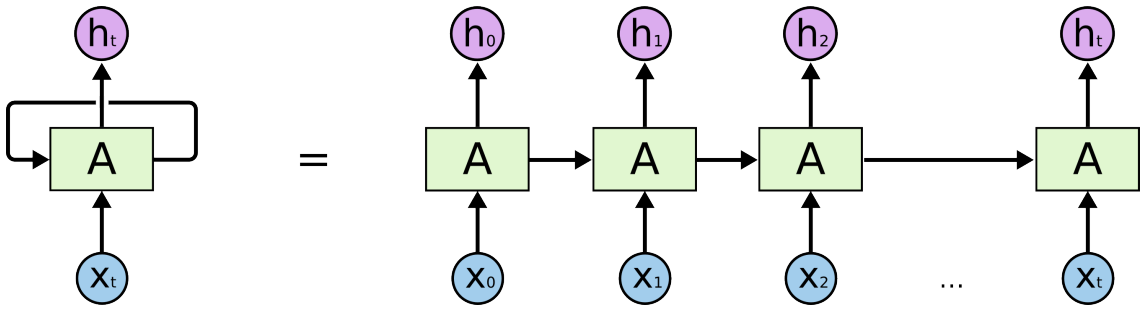
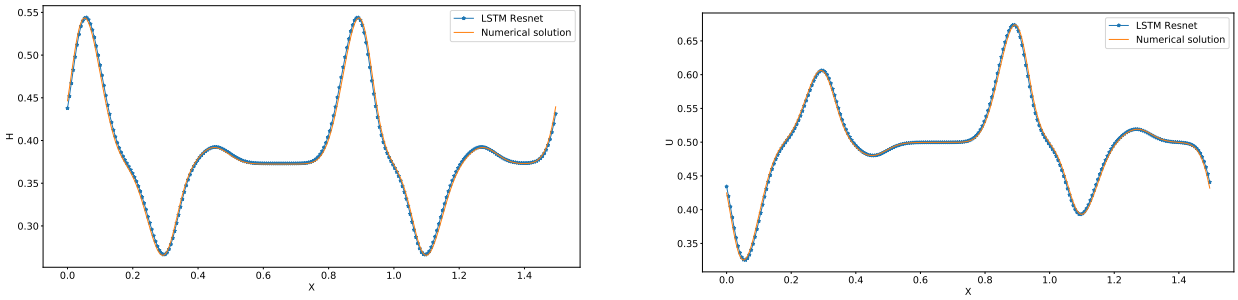


Figure 5.1: RNN unrolled in time.

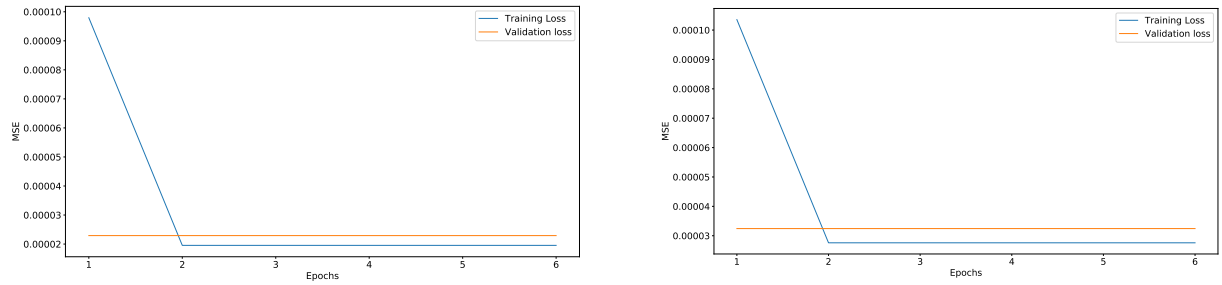
There are many RNN variants, we will use a long short-term memory (LSTM). An LSTM has an advantage over other types of RNNs of being able to learn long-term dependencies in the data as well as short-term ones. We note that we have modified the RNN architecture by adding the same skip connections that were in the residual feedforward architecture making this a residual LSTM network. All of the reasons for the skip connections, as explained in the previous section, still apply here, which is why we use them again.



(a) Height (b) Velocity

Figure 5.2: Predictions of LSTM Network

The output of the residual LSTM network is as well behaved and as accurate as the output from the residual feedforward network. Note the errors converge to a minimum much faster than the residual feedforward network. In fact the minimum for the training loss is arrived at after only two epochs compared to around five epochs for the residual feedforward network which indicates that this network is more efficient at learning than the residual feedforward network. This is further proof of the suitability of an RNN for this problem.



(a) Height (b) Velocity

Figure 5.3: Training and Validation losses

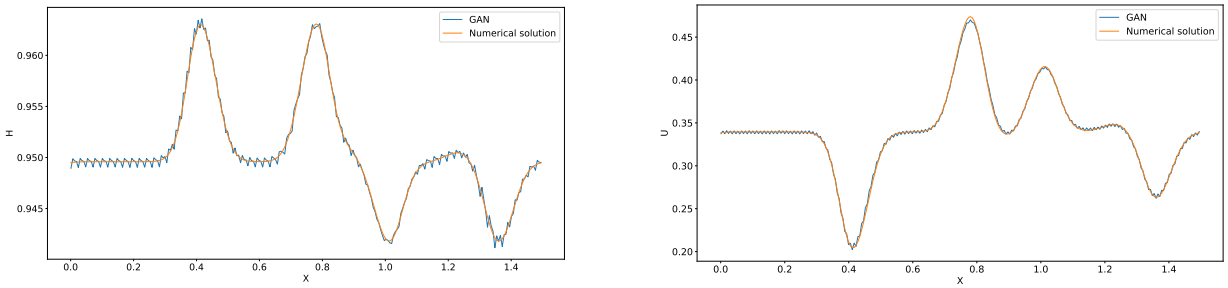
CHAPTER 6

GENERATIVE ADVERSARIAL NETWORK (GAN)

Another approach we can take is to treat each time step as a video frame and turn this into a frame prediction problem. This approach is by no means a novel one, see [13] developed the GAN architecture as a means of learning the mapping between input and output images and it has since been used for, among many applications, Image-to-Image translation, see [14], and video frame generation, see [15].

A GAN is a setup of two neural networks that work in tandem: a generator network and discriminator network. The role of the generator is akin to that of a master forger whose job it is to generate a fake piece of work with high enough quality that it can pass as an original. The discriminator's job is to determine whether or not and to what extent the input it receives is fake. During training the generator learns to produce more convincing fakes and the discriminator learns to distinguish more accurately. Training goes back and forth between both networks until we have a generator that can accurately map input images to output images.

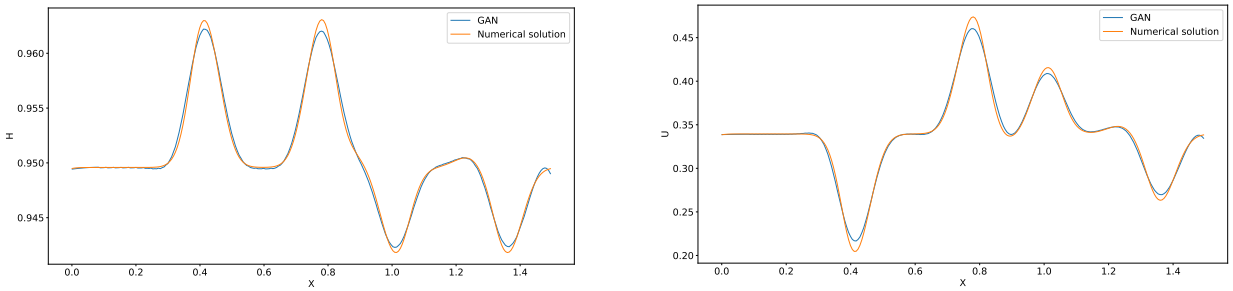
The GAN architecture we selected is based on the "pix2pix" network as in [15]. The Generator employs skip connections, identical to those in the residual networks, following the architecture of the "U-net", see [16]. These skip connections allow low level information to be moved from the top of the network to the bottom of it.



(a) Height (b) Velocity

Figure 6.1: Predictions of GAN Network

As we can see in the output signal, all the main features of the input signal are being picked up implying that the physical laws are being captured by the GAN as desired. What is undesirable is the high amount of noise present in the output. We speculate this noise to be due to the spectral bias theorem, see [18], which states that NNs have a harder time learning higher frequencies as compared to lower frequencies. We can deal with this by post-processing the output by applying a smoothing filter. We selected the Savitzky–Golay filter (SGF), see [17], and have found that it's use has significantly improved the signal quality.



(a) Height (b) Velocity

Figure 6.2: Predictions of GAN with SGF applied

We must mention that we have modified the training and testing datasets for the GAN. The GAN was displaying some divergent properties at the borders of the domain so we have extended the domain by copying over the wavefronts periodically; basically extending the waves periodically. After training has concluded the output is snipped back to the original domain. This has the result of keeping the areas of concern well outside of the domain we make our predictions on.

CHAPTER 7

CONCLUSION

In this work, we have tested the suitability of several well established NN architectures for solving the SWEs. Table 7.1 contains a summary of the results of each NN by listing the MSE for both H and U. These numbers we obtained on the testing dataset and are the averages of five runs of the network. The hyperparameters of every network were kept identical for each run with the only change being the training dataset. The data is generated using random ICs some of which each present varying degrees of difficulty for the network to learn. We average over different datasets to ensure that our measurements are not biased towards or against the particular waveforms present in each dataset.

Network	MSE H	MSE U
Feedforward	$4.35e{-4}$	$8.17e{-4}$
Residual	$3.59e{-5}$	$2.84e{-5}$
LSTM	$1.73e{-5}$	$2.32e{-5}$

Table 7.1: Summary of performances

The results show a trend of increased performance as the networks become more complex. The least complex network is the feedforward network and is, unsurprisingly, the network with the poorest performance. The feedforward network gains quite a performance boost with the addition of the skip connections in the feedforward residual network signifying that the skip connections are a very important aspect to consider in constructing a network suited for this problem.

Going from a feedforward architecture to a recurrent architecture in the LSTM network provides another increase in performance although it is not as sizable as the performance increase that came with adding the skip connections. However, the LSTM network requires a lot less training time to achieve its best performance and so is the most efficient so far.

We did not include the GANs result in here for a number of reasons. Firstly, the loss function is

not as instructive for the GAN as for the other networks each with a well defined loss function and besides, the discriminator network acts as the generators loss function and so comparing it's output to the other networks that have a well defined loss function for each parameter is unfair. The GAN also benefits from having the domain of the training data extended periodically so, again, it would not be fair to compare it to the other networks that do not share this advantage.

The GAN's output is at this point unusable due to the high amount of noise plaguing its output. The noise was reduced to a large extent by the SGF but even with the filter applied the results are not as good as the LSTM or even residual feedforward networks. We must also mention how inefficient and difficult training this network is which is due to it's complex design. The GAN on average took one hundred times longer to train compared to the LSTM and was the most sensitive to hyperparameter tuning where the slightest change in hyperparameters resulted in drastic changes to the output indicating that it is not at all stable.

It is for those reasons that we believe the LSTM network to be the most suited for the task at hand. It has the advantage of being the only network that is designed to handle time series data and our results indicate that it is the most accurate and efficient network of the bunch.

In future works, we aim to assess these networks© performance with predicting multiple time steps in advance which will allow us to investigate the stability of these networks. We also hope to be able to implement a NN to solve the two dimensional SWEs which is likely to be a much more complex undertaking.

REFERENCES

[1] A. Geyer and R. Quirchmayr, "Shallow water equations for equatorial tsunami waves," *Philisophical Transactions of the Royal Society A*, vol. 376, 2111 2018. DOI: 10.1098/rsta.2017.0100.

[2] S. Barman, D. Johnson, and B. S, "Application of the Shallow Water Equation to Real Flooding Case," *International Journal of Advanced Science and Technology*, vol. 29, pp. 1169–1174, 2 2020.

[3] A. Bihlo, "A generative adversarial network approach to (ensemble) weather prediction," *Neural Networks*, vol. 139, pp. 1–16, 2021. DOI: https://doi.org/10.1016/j.neunet.2021.02.003.

[4] K. S. Justin Sirignano, "DGM: A deep learning algorithm for solving partial differential equations," 2017. arXiv: 1708.07469 [q-fin.MF].

[5] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[6] J. S. Denker, W. R. Gardner, H. P. Graf, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, H. S. Baird, and I. Guyon, "Neural Network Recognizer for Handwritten Zip Code Images," 1989.

[7] W. Zhang, K. Itoh, J. Tanida, and Y. Ichioka, "Parallel distributed processing model with local space-invariant interconnections and its optical architecture," *Applied Optics*, vol. 29, pp. 4790–4797, 32 1990.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[9] P. Lax, "Weak solutions of nonlinear hyperbolic equations and their numerical approximation," *Communications on Pure and Applied Mathematics*, vol. 7, pp. 159–193, 1954.

[10] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007, http://numerical.recipes/, ISBN: 0521880688.

[11] R. Reed and R. J. Marks, "Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks," 1998.

[12] S. R. J. S. Kaiming He Xiangyu Zhang, "Deep Residual Learning for Image Recognition," 2015. arXiv: 1512.03385 [cs.CV].

[13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," 2014. arXiv: 1406.2661 [Stat.ML].

[14] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," 2016. arXiv: 1611.07004 [cs.CV].

[15] W. Xiong, W. Luo, L. Ma, W. Liu, and J. Luo, "Learning to Generate Time-Lapse Videos Using Multi-Stage Dynamic Generative Adversarial Networks," 2017. arXiv: 1709.07592 [cs.CV].

[16] P. F. Olaf Ronneberger and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," 2015. arXiv: 1505.04597 [cs.CV].

[17] A. Savitzky and M. J. E. Golay, "Smoothing and Differentiation of Data by Simplified Least Squares Procedures," *Analytical Chemistry*, vol. 36, pp. 1627–1639, 6 1964.