NTNU
Norwegian University of
Science and Technology

Halvor Snersrud Gustad

Using Artificial Neural Networks for Predicting Bending Moments of Riser Structures

June 2019

NTNU
Norwegian University of
Science and Technology

Norwegian University of
Science and Technology

Using Artificial Neural Networks for Predicting Bending Moments of Riser Structures

Halvor Snersrud Gustad

Applied Physics and Mathematics
Submission date: June 2019
Supervisor: Elena Celledoni
Co-supervisor: Per Thomas Moe

Norwegian University of Science and Technology
Department of Mathematical Sciences

Summary

Structural fatigue was until recently, not considered a challenge for riser and wellhead systems. Therefore, design codes did not include verification of structural loads. Further development of monitoring systems and numerical modeling techniques have however revealed that semi-submersible rigs with blowout preventer, and marine riser systems may cause significant wear and tear on permanently installed equipment.

Being able to calculate the bending moments of the riser and wellhead system is important in order to calculate the accumulated fatigue of the system. This may again be used to calculate the expected remaining life time of the system. In this thesis we investigate the possibility of using artificial neural networks to calculate the bending moments of a simulated riser structure, based on a limited number of sensors.

A numerical solver for a given set of governing riser equations found in the literature about riser mechanics is proposed. The choices of discretization methods in space and time is explained. The numerical solver is later used to create training and test data for the artificial neural networks.

The theory of treating deep artificial neural networks as a discretization of dynamical systems, based on recently published articles [HR17, CCHC19, E17], was studied. Three different artificial neural network architectures were implemented based on the articles in addition to a more traditional artificial neural network.

The networks performance was evaluated based on their ability to predict on unknown simulated riser data with and without added noise. The dynamical system inspired networks showed better properties when predicting on noisy data than the more traditional one. The networks were able to predict the bending moments of the riser with a high accuracy. The number of sensors and their positioning were found to be of importance to the accuracy of the artificial neural networks.

Sammendrag

Tretthet var inntil nylig ikke ansett som en utfordring for stigerør- og brønnhodesystemer. Belastning på strukturene ble derfor ikke tatt med i design kravene. Siden har utvikling av numeriske modeller og overvåkningssystemer vist at flyterigger med BOP (blowout preventer) og stigerørsystemer kan forårsake signifikante laster på permanent installert utstyr. Dette vil igjen føre til slitasje.

Ved å kunne kalkulere bøyemomentene stigerøret og brønnhodesystemet opplever kan man finne ut hvor slitt systemet er. Dette kan igjen bli brukt til å kalkulere den forventede resterende levetiden til systemet. I denne masteroppgaven ser vi på muligheten til å bruke kunstige nevrale nettverk til å predikere bøyemomentene til et simulert stigerør basert på en begrenset mengde sensorer.

En numerisk løser for et gitt sett ligninger som beskriver stigerøret, hentet fra litteraturen om stigerørsmekanikk, er foreslått. Valgene av diskretisering i rom og tid er drøftet. Den numeriske løseren blir siden brukt til å produsere trenings- og testdata for de kunstige nevrale nettverkene.

Teori om å behandle dype kunstige nevrale nettverk som diskretiseringer av dynamiske systemer, basert på nylig publiserte artikler [HR17, CCHC19, E17], ble undersøkt. Ut ifra på denne teorien ble tre kunstige nevrale nettverk studert videre og implementert i tillegg til et mer tradisjonelt nettverk.

Nettverkenes prestasjoner ble evaluert ut ifra deres evner til å predikere basert på ukjent simulert stigerørdata og simulert stigerørdata med støy. De dynamisk system inspirerte nettverkene viste bedre egenskaper når det gjaldt å predikere på data med støy enn det tradisjonelle nettverket. Nettverkene predikerte bøyemomentene til det simulerte stigerøret med høy nøyaktighet. Antall sensorer og deres posisjon viste seg også å være av betydning for nøyaktighetene til nettverkene.

Preface

This thesis is written as the final part of my 5 year master's degree programme at the Norwegian University of Science and Technology (NTNU) in Applied Physics and Mathematics, with specialization in the field of Industrial Mathematics. The thesis was written for the course TMA4900 - Industrial Mathematics, Master's Thesis, in the spring of 2019. The thesis is partially inspired by my specialization project written in the fall of 2018, which included a study of simulating riser behaviour.

This project has been supervised by professor Elena Celledoni at NTNU and Per Thomas Moe at TechnipFMC. Great thanks goes to Elena for great guidance and discussions. The hours spent with you in your office and the proofreading you have done on the thesis have been invaluable.

My gratitude also goes to Per Thomas for sharing his knowledge about riser analysis and answering my many questions and requests. I have valued the meetings with you and the colleagues you have invited.

Thanks also goes out to Nikita Kopylov for using his knowledge in the field of machine learning to help me understand necessary parts of the theory and for joining in on the discussions at Elena's office. I would also like to thank Henrik Sperre Sundklakk for using his expertise in numerical mathematics to assist with the convergence analysis concerning the numerical solution to the governing riser equations. My appreciation also goes out to Per Gustafsson and Alexander Ulanov at TechnipFMC for frequently sharing their wisdom on riser- and wellhead analysis in the meetings set up by Per Thomas.

Halvor Snersrud Gustad, June 2019, Trondheim

Contents

Summary i

Sammendrag iii

Preface v

Table of Contents viii

List of Tables x

List of Figures xii

1 Introduction 1

2 Literature Review 3

3 Theory 5
3.1 Riser . . . 5
3.1.1 Riser equations . . . 5
3.1.2 Airy wave theory . . . 6
3.1.3 Simulations . . . 6
3.2 Artificial Neural networks . . . 9
3.2.1 Feedforward Neural Networks . . . 11
3.2.2 Residual Networks . . . 13
3.2.3 Recurrent Neural Networks . . . 16
3.3 Optimization . . . 24
3.3.1 Gradient descent . . . 25
3.3.2 Newton method . . . 26
3.3.3 Quasi-Newton method . . . 26
3.3.4 Gauss-Newton method . . . 29
3.3.5 L-BFGS . . . 31
3.3.6 Stochastic descent . . . 32

4 Experiments 35
4.1 Riser simulation . . . 35
4.1.1 Initial setup . . . 35
4.1.2 Spatial discretization . . . 36
4.1.3 Time integration . . . 37
4.2 RNN Models . . . 38
4.2.1 Effects of Δt . . . 40
4.2.2 One sensor different positions . . . 45
4.2.3 Several sensors at different positions . . . 48
4.2.4 Diffusion constant . . . 51
4.2.5 Abilities of regularizers . . . 55
4.2.6 Optimization . . . 56

5 Discussion 61
5.1 The recurrent neural networks . . . 62
5.1.1 Structure . . . 62
5.1.2 Hyper parameters . . . 62
5.2 Avoiding overfitting . . . 63
5.2.1 Regularizer . . . 63
5.2.2 Early stopping . . . 63
5.2.3 Pruning . . . 64
5.3 Training . . . 64
5.3.1 Parallel implementation . . . 64
5.3.2 Coding language . . . 65

6 Conclusion 67

Bibliography 69

List of Tables

4.1 Parameters for the simulations 35
4.2 The five different types of waves used in the training and test sets. 36
4.3 R^2 values for the four different models with step size $\Delta t = \frac{1}{N}$ where N is the number of layers in the RNNs. The Vanilla columns are a reference columns and has no dependency on Δt, but is added to compare the new models to the original one. Noise*: Standard deviation = 0.1. Noise**: Standard deviation = 0.01. 41
4.4 R squared values for the four different models with step size $\Delta t = 1$. The Vanilla columns are a reference columns and has no dependency on Δt, but is added to compare the new models to the original one. Noise*: Standard deviation = 0.1. Noise**: Standard deviation = 0.01. 42
4.5 R^2 values with one sensor on the riser measuring displacement. Noise*: Training set with noise. The standard deviation of the noise was set to 0.01. 46
4.6 R^2 values for one sensor on top measuring inclination and one sensor somewhere else on the riser measuring displacement. Noise*: Training set with noise. The standard deviation of the noise was set to 0.01. 47
4.7 Chebyshev 3 50
4.8 Chebyshev 5 50
4.9 Chebyshev 10 50
4.10 Legendre 3 50
4.11 Legendre 5 50
4.12 Legendre 10 50
4.13 Equidistant 3 51
4.14 Equidistant 5 51
4.15 Equidistant 10 51
4.16 Output with negative diffusion constants, the two top rows represent the training set with added noise. Noise*: Standard deviation = 0.1. Noise**: Standard deviation = 0.01. 53

4.17 Output with no diffusion constant, the two top rows represent the training set with added noise. Noise*: Standard deviation = 0.1. Noise**: Standard deviation = 0.01. 53
4.18 Output with positive diffusion constants, the two top rows represent the training set with added noise. Noise*: Standard deviation = 0.1. Noise**: Standard deviation = 0.01. 54
4.19 Dynamical system based regularizer (3.37), the two top rows represent the training set with added noise. Noise*: Standard deviation = 0.1. Noise**: Standard deviation = 0.01. 55
4.20 Weight decay (3.36), the two top rows represent the training set with added noise. Noise*: Standard deviation = 0.1. Noise**: Standard deviation = 0.01. 55
4.21 No regularizer, the two top rows represent the training set with added noise. Noise*: Standard deviation = 0.1. Noise**: Standard deviation = 0.01. 56

List of Figures

3.1 Figure with different activation functions. Note the different y-axes. . . . 9
3.2 Neural network. Yellow: input layer, green: hidden layers, red: output layer. The edges between the nodes represents weights. 10
3.3 Compact illustration of a FNN first illustrated in figure 3.2. Yellow: input layers, green: hidden layers, red: output layer 17
3.4 A compact illustration of a RNN with 4 hidden layers. Yellow: input layers, green: hidden layers, red: output layer. The yellow and the green bullet to the very left in the figure will be considered the first input layer and the first hidden layer, respectively. The second most left bullets will be considered the second input layer and the second hidden layer and so on. 17

4.1 The Hermite shape functions on the standard interval $[-1, 1]$ 37
4.2 The blue line represents the current profile of the waves represented by Airy wave theory at maximum peak. Observe the different first axis for all the figures. 38
4.3 Adding waves and keeping the other parameters constant. The blue curve represents the riser. The waves correspond to the 8.5m waves in figure 4.2. 38
4.4 Bending moments for a riser of a hundred meters. The largest bending moments can be found where the riser is clamped to the sea bed. 40
4.5 Anti-symmetric RNN for predicting riser bending moments with sampling frequency of 100 Hz and using the last 50 states to make predictions. Top figure is the model's prediction on the training and test set with $\Delta t = 1/N$ and no noise. In the middle figure noise has been added and in the bottom figure the same noise has been added to the same model with $\Delta t = 1$ . . . 43
4.6 Standard RNN for predicting riser bending moments with sampling frequency of 100 Hz and using the last 50 states to make predictions. Top figure is the model's prediction on the training and test set with $\Delta t = 1/N$ and no noise. In the middle figure noise has been added and in the bottom figure the same noise has been added to the same model with $\Delta t = 1$ . . . 44

4.7 The value of the cost functions for the different RNNs as a function of time. Left: $\Delta t = 1$, right $\Delta t = \frac{1}{N}$. We observe that models with $\Delta t = 1$ were able to reach a lower value for the cost function before convergence. 45
4.8 The cost function on the training set as a function of the distance from the displacement sensor to the rig. The blue curve is the error on the training set without noise and the orange curve is the error on the training set with noise. 47
4.9 The cost function on the training set as a function of the distance from the displacement sensor to the rig. The blue curve is the error on the training set without noise and the orange curve is the error on the training set with noise. 48
4.10 Positions for the displacement sensors for the different settings concerning number of sensors and their positions. 49
4.11 Phase portraits for the underlying dynamical system of ODEs belonging to a Standard RNN initiated with transformation matrices with either positive (t.l) negative (t.r.) imaginary (b.l.) or imaginary with diffusion (b.r.) eigenvalues 52
4.12 Training time for the different diffusion settings, the legend gives information about the value of γ. 54
4.13 The figures show how the cost function is reduced as a function of time during training with the different regularizers. The figure to the left uses the custom regularizer for the dynamical system based neural networks. The middle figure uses no regularizer. The figure to the right uses the standard weight decay regularizer. 56
4.14 The figures shows the value of the cost function as a function of training time. Four of the optimization algorithms explained in section 3.3 were used to train the Anti-Symmetric RNN with four different depths. The number of layers are given by the figure titles. 57
4.15 The figures show how the value of the error function decreases as a function of time (top row) and iterations (bottom row) for RNNs with 20, 50 and 100 layers. Observe the curves in the figures are different in time, but essentially the same in iterations. Observe that the curves in the figures in the lower row are not identical, this is due to rounding errors of the numerical solver of the standard BFGS matrix. 58
4.16 The figure shows the time spent for completing 1000 iterations using the standard BFGS method and the BFGS method with Cholesky decomposition for training different Anti-Symmetric RNN. Observe how the blue curve increases faster with the number of layers than the orange curve. . . 59

Chapter 1

Introduction

A riser is a conduit connecting the oil rig to the subsea wellhead. The wellhead is fixed to the seabed, and allows for very little to no movement. The riser oscillates back and forth due to its long slender structure, these oscillations are one of the main reasons for fatigue on the riser and wellhead system. Due to the wellhead being immovable, the largest fatigue is expected to be found at the connection between the wellhead and the riser. This is comparable to an oscillating fixed beam, for which the largest stress values are found where it is fixed.

Currently, the expected lifetime for the riser system is calculated using methods which underestimates the true lifetime. This is because failure of the system could be disastrous. Being able to calculate the accumulated fatigue of the riser system means we can predict how long the system safely can stay in production. In other words, estimate the riser systems remaining lifetime based on its stress cycles. By doing so the riser system can stay in production closer to its actual life time.

In this thesis we therefore consider a mathematical model of a one-dimensional riser system. The model consists of a partial differential equation which is solved using a set of boundary conditions and different forcing terms based on which sea state we wish to simulate. The equations were obtained from literature about riser mechanics. The mathematical model was set up in collaboration with TechnipFMC.

We solve the riser equations numerically for different sea conditions due to different wave effects. The information about the risers displacement, inclination and curvature produced by the numerical solver will be stored. The accumulated fatigue can be calculated based on the bending moments the riser has experienced through its lifetime, which in turn is calculated from the curvature it undergoes. Being able to calculate the curvature cheaply will therefore be valuable for the offshore oil industry.

Several types of artificial neural networks will be trained based on the information

about the displacement, inclination and curvature of the riser and the results will be presented. Traditional artificial neural networks will be introduced and more recent networks will be explained based on the traditional ones. Their forward propagation algorithms will be derived together with the corresponding algorithms for finding the gradients.

Different optimization algorithms for training the artificial neural networks will be introduced based in the existing literature on optimization. Their properties are later discussed along with their performance when training the artificial neural networks. General optimization algorithms such as gradient and Newton methods are considered together with tailored stochastic descent algorithms.

A selection of artificial neural networks will be trained to predict the bending moments to the riser by the wellhead based on a set of sensors. Sensors are usually expensive and hard to maintain when fixed to a riser and lowered several meters bellow the ocean surface. In certain cases the depths can be as large as $3000\,\mathrm{m}$. This makes it attractive to only use a few sensors as close to the surface as possible. The different networks performance will be evaluated and discussed along with a discussion on the number and positioning of sensors. The performance of the given networks will be measured in how accurate their predictions are on the data sets created by the numerical simulator along with how well they handle noise.

The outline of the thesis goes as follows. Chapter 2 starts with a short introduction to the existing literature on riser modeling as well as the literature about artificial neural networks. Chapter 3 first gives a brief introduction concerning the governing equations of the riser and the discretization for the numerical solver. An in depth introduction to this theme can be found in [Gus19b, Gus19a]. After the introduction to the governing riser equations the theory of the different artificial neural networks is presented. At the end of chapter 3 different types of optimization algorithms are introduced. Chapter 4 includes results and discussion from experimenting with different artificial neural networks, sensors settings and optimization algorithms. Chapter 5 discuss the overall results as well as possible improvements to the current methods. Lastly a conclusion is found in chapter 6.

Chapter 2

Literature Review

Riser literature

Calculating the properties of a riser structure undergoing external forces is not a new phenomenon. In 1950 J.R. Morison and his team introduced a set of equations for calculating the force acting on a slender structure submerged in a moving fluid [MOJS50].

Later P.J O'Brien developed a finite element method for non-linear movement of offshore structures in three dimension [OMD87]. The method used Morison's equations in three dimensions. In 1989 O'Brien developed a finite element method using hybrid beam elements for simulating flexible risers in three dimensions [OM89].

In 1983 Flexcom, a software package for performing finite element analysis on offshore structures, was released. This software package along with packages such as RIFLEX and OrcaFlex have become important tools within the field of riser analysis.

Finite difference methods can also be utilized when analyzing a riser structure, but finite elements are far more popular. The reason for such low interest is explained in [PS95]. They argue that finite element methods are preferred due to better stability properties and simple implementation of boundary conditions.

Several variations of the finite element method have been developed and tailored to different problems. One such method is the rigid finite element method (RFEM). Later, modifications to the RFEM have been made to easier analyze bending and torsion. Boundary conditions, including for instance friction in joints, can with this method be considered without much effort [AWBD15].

Artificial neural network literature

The first mathematical model of an artificial neural network, which is considered to be a feed forward neural network, is credited to Nicolas Rashevsky in 1936 [RN10] and was brought further with the work of McCulloch and Pitts [MP43, CS88]. McCulloch and Pitts' idea was to use an activation function within neurons with a hard threshold, meaning it only passed on a signal if the input values were sufficiently large. These types of neurons become known as perceptrons. In their paper McCulloch and Pitts only used one layer,

but argued that any desired functionality would be possible by connecting several layers.

After this, new types of artificial neural networks were developed. One important neural network credited John Hopfield [Hop82] and David Rumelhart [RHW86] is the recurrent neural network. This type of network has the ability to respect the sequentially of the data and consequently has many types of applications. A further development of such networks was done by Sepp Hochreiter and Jürgen Schmidhuber in 1997. They introduced a method known as Long Short Term Memory (LSTM) [HS97].

After the field of artificial neural networks was first introduced it has evolved tremendously. The evolution of the neural networks has been correlated with the available processing power [RN10] and has in the recent decades had large breakthroughs. A reoccurring problem within the artificial neural network community is the vanishing gradient problem and its difficulty was proved by Yoshua Bengio et. al. [BSF94] and is still being studied extensively [PMB12, PMB13].

Despite the troubles with the vanishing gradient, neural networks are one of the most popular machine learning algorithms. Much effort has been made to train these methods as fast and as good as possible. Deterministic optimization algorithms can be used, but a more popular choice are stochastic optimization methods such as the stochastic gradient descent [Bot10]. Stochastic optimization methods can outperform deterministic methods when training neural networks with large data sets and research on taking good steps is an ongoing process. Recently, methods such as AdaGrad [CDHS11], Adam [KB15] and Nadam [Doz16] has proven to work well for certain neural networks if the right hyper parameters are chosen. Other stochastic versions of well known second order optimization algorithms has shown good results, such as the stochastic Levenberg-Marquardt algorithm [LBOM98].

Recently, papers have been published attempting to solve the training problems by considering certain neural networks as a discretization of a dynamical system [HR17, CCHC19, E17] or viewing the minimization of the error function as an optimal control problem [BCE+19]. These types of analogies are still being in studied, but are showing promising results when it comes to training and predicting properties.

In addition to the vanishing gradient problem, artificial neural networks are also suffering from the well known problem of overfitting. Common statistical methods such regularizers are often used in artificial neural networks. The possibly most used regularizer is known as weight decay is easy to implement and gives good results [KH92]. Other methods that can be applied on top of weight decay have been developed and methods such as dropout [SHK+14] and the winning lottery ticket [FC18] are based on pruning the neural networks and have shown good results e.g. [KSH12].

Chapter 3

Theory

3.1 Riser

3.1.1 Riser equations

The riser equations can be found using energy conservation. This being the kinetic energy as well as the potential energy due bending and strain. Since the riser is lowered in water it will also experience the effects of damping. Combing this information we can derive the following equations

$$EI\frac{\partial^4 u(x,t)}{\partial x^4} - T\frac{\partial^2 u(x,t)}{\partial x^2} + \rho\frac{\partial^2 u(x,t)}{\partial t^2} + c\frac{\partial u(x,t)}{\partial t} = f(x,t). \tag{3.1}$$

The boundary conditions of the riser will be based in the riser being fastened to the well-head at the seabed with no joint. The riser will be hanging freely from the oil rig and the oil rig will be not be set into motion. This will lead to the following set of boundary conditions

$$\begin{array}{rclrcl} u(0,t) &=& 0, & u(L,t) &=& 0 \\ u''(0,t) &=& 0, & u'(L,t) &=& 0 \end{array} \tag{3.2}$$

where L is the considered to the be distance from the surface to the seabed. The complete derivations of the PDE (3.1) and its boundary conditions can be found in [HGHC14].

The right hand side of (3.1) is the forcing term of the equation and is assumed to be a function of the water velocity due to waves and ocean current. This a simplification since also vortex induced vibrations (VIV) will occur in real situations. To include VIV a three dimensional model must be evaluated with a significant increase in the complexity of the simulations and this was not attempted in this thesis.

To calculate the force on the riser due to the velocity of the water around it the relative velocity must be considered. This was done by Morison and his team in 1950 [MOJS50] and later adjusted for cylinders in water [Spa07]. The hydrodynamic forces in the riser can be divided into two terms, one for drag and one for inertia

$$f = f_d + f_I.$$

For cylinders in orthogonal flow we have the following relations

$$\begin{aligned} f_d &= \frac{1}{2}\rho_w C_D R(v-\dot{u})|v-\dot{u}|, \\ f_I &= C_M \rho_w A_e \dot{v} - (C_M - 1)\rho_w A_e \ddot{u}, \end{aligned} \tag{3.3}$$

where ρ_w is the water density, R is the radius of the riser, v is the speed of the flow and $\dot{u}$ is the velocity of the riser orthogonal to its axis. The constants C_M and C_D are the inertia and drag coefficients, respectively and A_e is the area of the external cross section.

If we insert (3.3) into (3.1) we can calculate the forcing term as

$$f = \frac{1}{2}\rho_w C_D R(v-\dot{u})|v-\dot{u}| + C_M \rho_w A_e \dot{v} - (C_M - 1)\rho_w A_e \ddot{u}. \tag{3.4}$$

Observe a non-linearity in the first term in (3.4). This non-linearity makes the simulation of the overall problem more challenging.

3.1.2 Airy wave theory

Airy wave theory gives a sufficient approximation to gravity dominated surface waves in a homogeneous fluid when the seabed is uniform. We assume the depths to be such that when scaled any changes in the seabed is insignificant and can therefore be considered uniform. The fluid velocity of a water particle in a wave is then given by

$$v(x,t) = \frac{H}{2}\frac{\cosh(\lambda(L-x))}{\sinh(\lambda L)}\omega\cos(\omega t) \tag{3.5}$$

where H is the wave amplitude, $\lambda = \frac{2\pi}{\text{wave length}}$, $\omega = \frac{2\pi}{\text{wave period}}$ and L is the length of the riser. [Spa07, p. 266][Gus19a]

3.1.3 Simulations

The riser system can be solved numerically by discretizing the governing equations using a finite element method in space and some form of time integration in time. Since the equations consist of a fourth order derivative then polynomials of order 2 or higher must be used. Additionally, if the solution is to be continuous in the second derivative, 3rd order polynomials or higher must be applied.

Finite element methods rely on finding the weak form of a PDE by integrating with a suitable test function. For the riser equations the weak form can be found by multiplying by a test function v and integrating by parts. By doing so the following equation is derived

$$\int_0^L EIu''v'' + Tu'v' + \rho\ddot{u}v + c\dot{u}v\,dx = \int_0^L f(x,t)v\,dx, \quad \forall v \in V, \tag{3.6}$$

where V is such that

$$V := \{v \in H^2([0,L]) : v(0) = v(L) = v'(L) = 0\}, \tag{3.7}$$

a Sobolev space of functions on $[0, L]$ see [Qua09] for details and definitions. The weak form can then be stated as

$$\text{find } u \in V: \qquad a(u,v) = F(v) \qquad \forall v \in V, \tag{3.8}$$

where the bilinear form

$$a(u,v) := \int_0^L EIu''v'' + Tu'v' + \rho\ddot{u}v + c\dot{u}v\,dx$$

and linear form

$$F(v) := \int_0^L f(x,t)v\,dx$$

are the left and right hand side of (3.6) respectively.

Lax-Milgram theorem guarantees existence and uniqueness of solutions of (3.8)

Theorem 3.1.1 (Lax-Milgram)**.** Given a Hilbert space V, a continuous and coercive[1] bilinear form $a(\cdot,\cdot) : V \times V \to \mathbb{R}$ and a linear continuous functional $F(\cdot) : V \to \mathbb{R}$. Then there exists a unique solution to (3.8)

see [Qua09, QV94, SM11, BS08] for proof.

If we for simplicity only consider the linear stationary equations, we have

$$\int_0^L EIu''v'' + Tu'v' + (\rho + c)uvdx = \int_0^L f(x)vdx. \tag{3.9}$$

Where the left hand side will be defined to be the bilinear form $a(u,v)$ and the right hand side is the linear form $F(v)$. To prove that $a(u,v)$ is coercive we use that

$$a(u,u) = \int_0^L EI(u'')^2 + T(u')^2 + (\rho + c)u^2dx. \tag{3.10}$$

Since the constants in (3.10) are positive we observe that $a(u,u)$ always increases with u, implying it is coercive with coercivity constant $\alpha = \min\{EI, T, (c+\rho)\}$.

To prove continuity of the bilinear form $a(u,v)$ we use the fact that if there exist a constant $M > 0$ such that

$$|a(u,v)| \leq M||u||_V||v||_V \qquad \forall u, v \in V \tag{3.11}$$

then $a(u,v)$ is continuous. [Qua09, p. 12]. Here V is the Sobolev norm corresponding to the space defined in (3.7). From here one may show that

$$|a(u,v)| \leq \left(EI + 2LT + 4L^2(\rho + c)\right)||u||_V||v||_V \tag{3.12}$$

where L is the length of the interval. Under appropriate restrictions on f, $F(\cdot)$ is also continuous and so (3.1) has a unique solution.

When replacing f with Morison equations the uniqueness analysis becomes quite involved and will not be explored further in this thesis.

[1] A bilinear form is coercive if for all $u \in V$ $a(u,u) \geq \alpha||u||_V^2$, [Qua09].

The weak form with Morison forcing terms included will change the right hand side of (3.6) to

$$F(z,u) = \frac{1}{2}\rho_w C_D R \int_0^L (v - \dot{u})\,|v - \dot{u}|\, z dx + C_M \rho_w A_e \int_0^L \dot{v} z dx. \tag{3.13}$$

By replacing the infinite dimensional function space, V, with a finite dimensional approximation, V_h, spanned by a set of piecewise polynomials then the discretized version of the weak form can the be defined as

$$\begin{aligned} &\sum_{j=0}^{N} \int_0^L EIu_j(t)\phi_j''\phi_i'' + Tu_j(t)\phi_j'\phi_i' + (C\ddot{u}_j(t) + c\dot{u}_j(t))\phi_j\phi_i\, dx \\ &= \rho_w \int_0^L \frac{1}{2} C_D R \left(v - \sum_{j=0}^{N} \dot{u}_j\phi_j \right) \left| v - \sum_{j=0}^{N} \dot{u}_j\phi_j \right| + C_M A_e \dot{v}\phi_i\, dx. \end{aligned} \tag{3.14}$$

where $C = \rho + (C_M - 1)\rho_w A_e$ and the rest of the parameters are as before. The previous equation can be rewritten as

$$\mathbf{Du}(t) + \mathbf{Au}(t) + \mathbf{M_p}\ddot{\mathbf{u}}(t) + \mathbf{M_c}\dot{\mathbf{u}}(t) = \mathbf{F}(t, \mathbf{u}). \tag{3.15}$$

where the ith element of F is given by the right hand side of (3.14).

This system of second order ordinary differential equations (ODEs) can be transformed into a system of first order ODEs, and may then be written on the form

$$\begin{bmatrix} \dot{\mathbf{u}}_\mathbf{0}(t) \\ \dot{\mathbf{u}}_\mathbf{1}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{M_p}^{-1}(\mathbf{A} + \mathbf{D}) & -\mathbf{M_p}^{-1}\mathbf{M_c} \end{bmatrix} \begin{bmatrix} \mathbf{u_0}(t) \\ \mathbf{u_1}(t) \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{M_p}^{-1}\mathbf{F}(t, \mathbf{u}) \end{bmatrix} \tag{3.16}$$

or more compactly

$$\dot{\mathbf{u}}(\mathbf{t}) - \mathbf{Bu} = \hat{\mathbf{F}}(t, \mathbf{u}), \qquad \mathbf{u}(0) = \mathbf{0}. \tag{3.17}$$

In (3.17), $\mathbf{B}$ is a linear stiff term and $\hat{\mathbf{F}}$ is a non-stiff non-linearity. Numerical time-integration techniques tailored to this special form of ODE systems have been considered by various authors, see [HO10, MW05] for a survey.

These methods are built by solving the linear part exactly using the matrix exponential of the linear term $\mathbf{B}$. For this reason, these methods are known as exponential integrators. This approach is quite effective for stiff problems. The simplest exponential integrator is the exponential Euler method

$$\mathbf{u_{n+1}} = e^{-h\mathbf{B}}\mathbf{u_n} + h\phi_1(h\mathbf{B})\hat{\mathbf{F}}(t_n, \mathbf{u_n}) \tag{3.18}$$

where $\phi_1(\mathbf{z}) = -e^{\mathbf{z}}\mathbf{z}^{-1}(e^{-\mathbf{z}} - \mathbf{I})$. This method is of order 1, but has improved stability compared to the classical Euler method [HO10, p. 223-225].

3.2 Artificial Neural networks

An artificial neural network, abbreviated *ANN*, consists of a set of nodes which is organized into different layers. The nodes in neighbouring layers are connected through a set of edges with different weights. These nodes attempt to replicate the function of neurons in the brain, firing when the electromagnetic signal reaches a certain threshold. The nodes in the ANN passes on a value based on the signals they receive and a previously defined activation function. It is common that the activation function is monotonically non-decreasing. The value of the received signal depends on the value passed by the previous nodes multiplied with a set of weights.

Common choices for the activation function involves, but are not limited to, the logistic function, the hyperbolic tangent function, ReLu and the identity function. It need not be the same for the different layers and it is not uncommon to choose the activation function in the last layer to be a different function from the others. This depends on what prediction one wishes to make. In this thesis $g(x)$ will denote the activation function in the last layer, often called the hypothesis function, while $\sigma(x)$ will denote the activation function in all the preceding layers.

ANNs can be used for both regression and classification of data. Regression implies that the output data is continuous, while classification means that the output data is discrete. It is common to use different activation and hypothesis functions when working with regression versus classification. The listed activation functions can be found in figure 3.1.

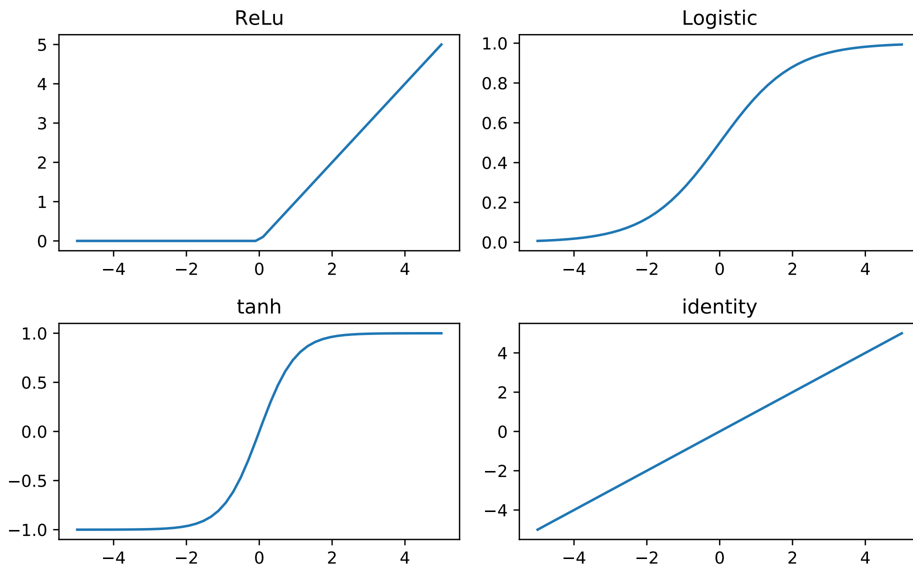

Figure 3.1: Figure with different activation functions. Note the different y-axes.

Within the machine learning community the term *feature* is used to define a single

measurable property of the data, the plural form *features* is therefore commonly used to define input data and is also done in this thesis. The term *labels* is used to define output data. The fact that machine learning often is used for classifications and labeling objects makes this a natural description of the output data. In this thesis we are working with a regression model, but the term *labels* will still be used to define output. These terms are from here on used in the same manner.

Any layer that is not an input or an output layer will be defined as a *hidden layer*. In this thesis each hidden layer will be defined as $a^{(k)}$ for $k = 1, \ldots, L$. The input layer is defined as $a^{(0)}$ and the output layer will be defined as Y^p where the superscript p is used to indicate that it is a prediction.

We will assume all layers to be dense, meaning that each node in one layer is connected to all the nodes in neighbouring layers by weighted edges. Figure 3.2 gives an illustration of a neural network with 4 hidden layers.

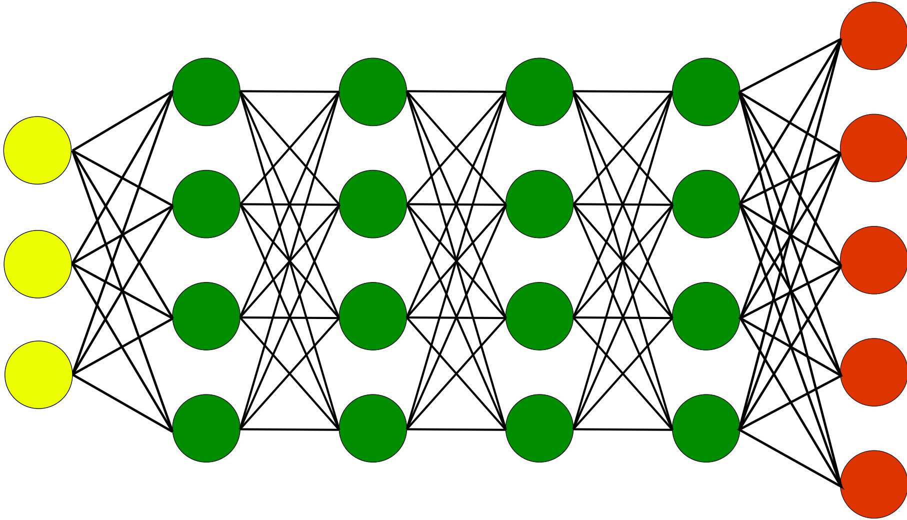

Figure 3.2: Neural network. Yellow: input layer, green: hidden layers, red: output layer. The edges between the nodes represents weights.

The goal when training any neural network is to minimize some predefined cost function of the training data [RN10]. Quite often this is set to be the Euclidean distance between the predictions on the training data and the true value of the labels. If one has extra information about the problem, or wishes to enforce some other types of qualities to the network, one can adjust the cost function accordingly [HR17, CCHC19, RPK17]. We will use a cost function defined as

$$E = \frac{1}{2} \sum_{l} ||Y_l^p - Y_l^t||_2^2 \tag{3.19}$$

where Y_l^p is the prediction on the lth sample in the training set and Y_l^t is the corresponding true value. If for instance the training set is a set of images then Y_l^p correspond to the

lth image. Y_l^p is parameterized by the weight matrices $W^{(k)}$ and bias vectors $b^{(k)}$ for $k = 0, \ldots, L$ where L is the number of hidden layers in the ANN.

When training an ANN we wish to minimize the error function (3.19) and therefore we get the following optimization problem

$$\min_{W^{(k)}, b^{(k)}} \frac{1}{2} \sum_l ||Y_l^p - Y_l^t||_2^2 \qquad \text{where } k = 0, \ldots, L. \tag{3.20}$$

It is also common to add a regularizer to reduce overfitting and, for some architectures, to uphold certain criteria for stability of the method. The general form of the error function will then be on the form

$$E = \frac{1}{2} \sum_{l=1}^{m} ||Y_l^p - Y_l^t||_2^2 + R(W)$$

where $R(\cdot)$ usually is a sum of norms.

Minimizing the error function (3.19) will reduce the error of the predictions on the training data, but this is not a guarantee that the network will make good predictions on unknown data. If the size of the network is too big it merely remembers the training data and will not be able to give good predictions even if the data is only slightly perturbed. This well known phenomenon known as overfitting is one of the larger issues when trying to implement a machine learning algorithm.

There are methods attempting to overcome the overfitting problem. One method is simply by trial and error, meaning to adjust the number and size of the layers until the ANN performs adequately [RN10]. However, there exists approaches to reduce the number of weights to avoid overfitting [RN10, KSH12]. Regularization techniques are used for reducing overfitting, but there also exists methods for pruning edges and nodes off the ANN [SHK$^+$14, FC18]. The pruning methods will be discussed in more detail in chapter 5.

3.2.1 Feedforward Neural Networks

A feedforward neural network (FNN) is one of the simplest neural networks and is widely used [BCE$^+$19]. The name *feedforward* neural network comes from how data being fed to the network is only evaluated by the activation function before it is passed forward to the next layer by the weighted edges. This differs from more sophisticated neural networks which will be introduced later.

The FNN does not take into account if the data passed to it is sequential. This means that for a neural network to make prediction based on time dependent sensor data it must be fed the data in a proper way. One possibility is to keep a list of the last N data points and calculate a prediction based on this list. Once a new data point is obtained the oldest data point in the list is removed and the next prediction can be made. Meaning we make predictions from features received from a window in time.

Propagation algorithms

In matrix form we may write the mathematical formula for the hidden layers and the output layer as

$$\begin{aligned} a^{(k)} &= \sigma(W^{(k-1)}a^{(k-1)} + b^{(k)}), \qquad k = 1, \dots, L \\ Y^p &= g(W^{(L)}a^{(L)} + b^{(L+1)}) \end{aligned}$$

where $a^{(k)}$ is the kth hidden layer, $W^{(k-1)}$ is the weight matrix between layer $k-1$ and k, $b^{(k)}$ is the bias in the kth layer and $\sigma(x)$ is some activation function applied component wise to the elements of x.

As mentioned previously the goal is to minimize a cost function such as (3.19). To find the minimizer we can use optimization algorithms utilizing the gradient of the cost function as given in (3.19). The gradient of cost function with respect to the parameters $W^{(k)}$ and $b^{(k)}$ for all the layers is usually found through the chain rule of differentiation. This process gives rise to an algorithm that in the machine learning context is known as backpropagation.

The backpropagation algorithm evaluates the gradient of the network for a given set of weights by first calculating the prediction of the training set. This prediction is used to evaluate the value of the error function. The partial derivatives of the error function is then found for the weights and biases in the last layer. Once these partial derivatives are found they are used to calculated partial derivative with respect to the weights and biases in the preceding layer. In this manner we propagate backwards until we have found all the partial derivatives of (3.19).

We can find the partial derivatives of the function (3.19) with respect to the components of $W^{(L-1)}$ by applying the chain rule

$$\begin{aligned} \frac{\partial E}{\partial W_{i,j}^{(L)}} &= \sum_l (Y_l^p - Y_l^t)^T \cdot \frac{\partial Y_l^p}{\partial W_{i,j}^{(L)}} \\ &= \sum_l (Y_l^p - Y_l^t)^T \cdot \frac{\partial g(W^{(L)}a_l^{(L)} + b^{(L+1)})}{\partial W_{i,j}^{(L)}} \\ &= \sum_l (Y_l^p - Y_l^t)^T g'(Z_l^{(L+1)}) \cdot e_i e_j^T a_l^{(L)} \\ &= \sum_l \left((Y^p - Y^t) \odot \sigma'(Z^{(L)})\right)_{il} \left(a^{(L)}\right)^T_{lj} \\ &= \left(\left((Y^p - Y^t) \odot \sigma'(Z^{(L)})\right) \left(a^{(L)}\right)^T\right)_{ij} = \left(\delta^{(L)} \left(a^{(L-1)}\right)^T\right)_{ij}. \end{aligned}$$

Finding the partial derivatives for the weights between the other layers is done in a similar fashion and using $\delta^{(L)}$ and the updating it. Note that the procedure is essentially the same for the partial derivatives of the biases.

Finding the minimizer of (3.19) may be done by previously designed minimization algorithms of first and second order or by stochastic descent algorithms. Each of these algorithms have different pros and cons and will be discussed further in section 3.3.

Drawbacks

Since we are applying a function such as the logistic or hyperbolic tangent function the information that is passed to the forward propagation may be squeezed until it is no longer noticeable. This is due to the fact that these function never give values larger than 1 or smaller than -1. This can make small, but significant changes in the features unable to affect the predictions. In other cases small disturbances in the input may have a huge effect on the output. This problem is known as the vanishing/exploding gradient problem.

In the case of FNNs a sufficient conditions can be derived for vanishing gradients [PMB13]. It can be shown that the partial derivatives for the error function for one training sample with respect to the weight matrix $W^{(L-k)}$ are

$$\frac{\partial E}{\partial W^{L-k}} = \left(\prod_{i=0}^{k} \sigma'(z_l^{L-k+i})(W^{L-k+i})^T \right)^T (a_l^L - Y_l^t)(a_l^{L-k-1})^T. \tag{3.21}$$

Using Cauchy-Schwarz inequality we can derive an upper bound for the norm of (3.21)

$$\begin{aligned} \left|\left| \frac{\partial E}{\partial (W^{L-k})^T} \right|\right|_2^2 &= \left|\left| \left(\prod_{i=0}^{k} \sigma'(z_l^{L-k+i})(W^{L-k+i})^T \right) (a_l^L - Y_l^t)(a_l^{L-k-1})^T \right|\right|_2^2 \\ &\leq \left|\left| \left(\prod_{i=0}^{k} \sigma'(z_l^{L-k+i})(W^{L-k+i})^T \right) \right|\right|_2^2 \left|\left|(a_l^L - Y_l^t)\right|\right|_2^2 \left|\left|(a_l^{L-k-1})^T\right|\right|_2^2 \\ &\leq \gamma^{2k} s^{2k} \left|\left|(a_l^L - Y_l^t)\right|\right|_2^2 \left|\left|(a_l^{L-k-1})^T\right|\right|_2^2 \end{aligned}$$

where γ is the largest value $\sigma'(x)$ takes and s^2 is the largest singular value of W^{L-k+i} for $i = 0, \ldots, k$. Similar computations can be found in [PMB12, BSF94].

It is therefore sufficient to say that if $\gamma s < 1$ then the norm of the partial derivative of the error function with respect to weight matrix W^{L-k} will decrease exponentially to zero with k. We see that the part of the gradient belonging to the weights far down in the system vanish due to the small factor γs and hence we have a vanishing gradient.

The above result is a sufficient conditions for vanishing gradients, if we reverse the calculations we can find a necessary condition for the exploding gradient problem. The necessary condition will be that the largest singular value must be larger than γ [PMB12].

3.2.2 Residual Networks

Since the first ANN was introduced by McCulloch and Pitts [CS88, MP43] there have been many attempts to improve the original version. Many variants have been implemented, each tailored for a specific field where they are to make predictions. Still, they all suffer to some degree with the problem of the vanishing/exploding gradient. One of the networks that are similar to a FNN, but has better training properties is the residual network (ResNet).

Forward propagation

The forward propagation for Haber and Ruthotto's [HR17] simplified version of residual networks is given by

$$a^{(k)} = a^{(k-1)} + \Delta t \sigma(W^{(k-1)} a^{(k-1)} + b^{(k)}) \tag{3.22}$$

$$Y^p = g(W^{(L)} a^{(L)} + b^{(L+1)}) \tag{3.23}$$

where Δt is some constant, usually 1, and the rest of the variables are the same as in section 3.2.1.

Even though this method of forward propagation has better properties than the feed forward propagation the vanishing gradient problem is still the cause for poorly trained networks. Forward propagation algorithms related to (3.22) and (3.23) have been studied in [HR17, E17]. One attempt to improve the gradient problem is the dynamical system approach.

Dynamical system approach

The dynamical system approach aims to solve the problem with the vanishing gradient for residual neural networks. The main concept is to view the network as a discretized version of a dynamical system [HR17, E17, CCHC19]. There already exists large amounts of mathematical theory about dynamical systems and their stability properties. If the ResNet can viewed as a dynamical system then this theory can be applied.

If we consider the network to be a discretization of a dynamical system, we see that (3.22) looks like a forward Euler discretization of the following system of ODEs

$$\dot{a}(t) = \sigma(W(t)a(t) + b(t)), \tag{3.24}$$

where $\sigma(\cdot)$ is applied element wise.

The neural networks ability to propagate information forward without loosing information (vanishing gradient) or making poor predictions due to noise (exploding gradient) can be evaluated by considering the stability of corresponding dynamical system [HR17, CCHC19, E17].

In their paper Haber and Ruthotto argue that if the real part of the eigenvalues of the Jacobian to (3.24) are negative then the minor changes in the input data will not affect the predictions and thus the forward propagation is stable and consequently should also be able to handle noise and generalize well. Even though the forward propagation is stable the learning problem stated as

$$\begin{aligned} &\min E(\mathbf{W}, \mathbf{b}) \\ \text{s.t. } &a^{(k)} = a^{(k-1)} + \Delta t \sigma(W^{(k-1)} a^{(k-1)} + b^{(k)}), \qquad k = 1, \dots, L \end{aligned} \tag{3.25}$$

will in this case be ill posed [HR17] making the backward propagation unstable. Another consequence of having negative eigenvalues is that valuable information from the input layer may disappear as the information propagates forward.

To ensure that the forward propagation is stable and the learning problem is well posed Haber and Ruthotto argue that the eigenvalues of the Jacobian should be purely imaginary.

This will preserve the gradient and data will flow through the whole network. As a consequence of the stable forward and backward propagation the modified neural network should be able to both handle noise and generalize will.

The Jacobian of (3.24) is given by

$$J(a) = W(t)\sigma'(W(t)a(t) + b(t))$$

where $\sigma'(x)$ is a diagonal matrix with x_i on the ith diagonal element. If $\sigma(x)$ is non-decreasing then $\sigma'(x)$ will always be diagonal positive semi-definite with real eigenvalues. In such cases the sign of the eigenvalues of the Jacobian will be the same as the eigenvalues of the weight matrix $W(t)$. By keeping the matrix anti-symmetric we can ensure that the system is always stable. This can be hardcoded in the ODE system by imposing

$$W(t) = C(t) - C(t)^T.$$

Essentially this means reducing the degrees of freedom of each matrix from n^2 to $\frac{n(n-1)}{2}$.

Another possibility is to change the forward propagation into a Hamiltonian system [HR17]. A Hamiltonian system has the form

$$\dot{y}(t) = \nabla_z H(y, z, t) \qquad \dot{z}(t) = -\nabla_y H(y, z, t)$$

and does in the autonomous case conserve the energy of the system [Asc08]. Here the term *energy* is the information flowing from the input layer to the output layer. An approach to creating a Hamiltonian system would be to set

$$H(y, z, t) = z(t)^T z(t) + f(y(t))$$

which gives us a first order system of the form

$$\dot{y}(t) = z \qquad \dot{z}(t) = -\nabla_y f(y). \tag{3.26}$$

If we let $\nabla_y f(y(t)) = -\sigma\left(W(t)y(t) + b(t)\right)$ then (3.26) can be rewritten as

$$\ddot{y}(t) = \sigma(W(t)y(t) + b(t)) \qquad y(0) = y_0, \quad \dot{y}(0) = \dot{y}_0.$$

This gives a system that preserves the energy in the Hamiltonian function H. There are however restrictions that need to be made on W such that the system does not suffer from vanishing/exploding gradients. To see how to do this one can go back to the first order system (3.26) and find the Jacobian of

$$\begin{bmatrix} \dot{y}(t) \\ \dot{z}(t) \end{bmatrix} = \begin{bmatrix} z(t) \\ \sigma(W(t)y(t) + b(t)) \end{bmatrix}.$$

The Jacobian is then

$$J\left(\begin{bmatrix} \dot{y}(t) \\ \dot{z}(t) \end{bmatrix}\right) = \begin{bmatrix} 0 & I \\ \sigma'(W(t)y(t) + b(t))W(t) & 0 \end{bmatrix}$$

where I is the identity matrix and $\sigma'(x)$ is a diagonal matrix with only positive entries as described before. The eigenvalues λ of the Jacobian can be found from the equation

$$\det(\lambda^2 I - \eta(t)W(t)) = 0$$

where $\eta(t) = \sigma'(W(t)y(t) + b(t))$.

As before we want the system to only have imaginary eigenvalues, and since $\eta(t)$ does not change the sign of the eigenvalues it is sufficient to restrict W to be negative definite. This can be enforced by design of the system by setting

$$W(t) = -C(t)^T C(t) \tag{3.27}$$

where $C(t)$ is any matrix [HR17]. In their paper [HR17] Haber and Ruthotto suggest using the leapfrog method for time integration. It is defined as

$$y_{n+1} = \begin{cases} 2y_n + h^2\sigma(Wy_n + b_n), & n = 0 \\ 2y_n - y_{n-1} + h^2\sigma(Wy_n + b_n), & n = 1, \dots, N-1 \end{cases}$$

for when discretizing the system into N time-steps. This method is sympletic and therefore has features like low error accumulation over long times and also yields an arbitrarily exact solution to a perturbed Hamiltonian system in finite precision [Asc08].

The local truncation error for this method given by

$$y(x_n + h) - y_{n+1} = \frac{h^4}{12} y''''(\eta), \tag{3.28}$$

which is an improvement to the local truncation error for Euler method given by

$$y(x_n + h) - y_{n+1} = \frac{h^2}{2} y''(\eta). \tag{3.29}$$

This indicates that the forward propagation of the Hamiltonian based neural networks will be more precise and needing fewer steps for the same accuracy in the solution. In this analogy, the number of steps in the discretization of the dynamical system is the number of layers in the neural network.

Due to the additional non-linearity of the weight matrices in the Hamiltonian based ANN the system will be harder to train [HR17]. This can be verified in section 4.2.1.

If the theory about the dynamical system should work and have good stability properties then the weight matrix $W(t)$ should change sufficiently slow. This can be enforced by adding a regularizer such that differences between the neighbouring weight matrices are penalized. This will be studied further in section 3.2.3.

3.2.3 Recurrent Neural Networks

When the sequence of the data is relevant for the prediction then architectures such as recurrent neural networks (RNNs) is better at making predictions than methods such as the FNN and the ResNets.

The main difference between recurrent neural networks and the previously described architectures is the way the information is passed to the network. In the earlier mentioned architectures all the information is passed into the network at the input layer and then propagated forward. In RNNs each layer is considered to correspond to a point in time and information is passed to this layer from the outside as well as from the previous hidden layer.

To give an intuitive figure illustrating the mechanics of the RNNs we first present a compact illustration of the FNN first shown in figure 3.2. This illustration can be found in figure 3.3.

Figure 3.3: Compact illustration of a FNN first illustrated in figure 3.2. Yellow: input layers, green: hidden layers, red: output layer

Here each bullet represents a whole layer of nodes. Figure 3.4 gives a compact illustration to how recurrent neural networks look like analogous to figure 3.3.

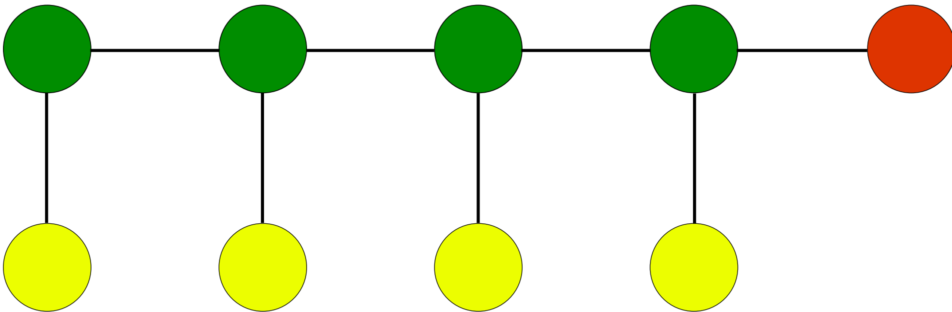

Figure 3.4: A compact illustration of a RNN with 4 hidden layers. Yellow: input layers, green: hidden layers, red: output layer. The yellow and the green bullet to the very left in the figure will be considered the first input layer and the first hidden layer, respectively. The second most left bullets will be considered the second input layer and the second hidden layer and so on.

The way the RNNs are constructed, adding new information to every layer, allow for the networks to keep a short term memory of the previous states. The number of layers in the RNNs decides how long the short term memory is.

Forward propagation

The forward propagation of the simplest recurrent neural network, popularly called *Vanilla RNN*, is given by the following equations

$$\begin{aligned} a^{(k)} &= \sigma(W^{(k-1)}a^{(k-1)} + V^{(k)}x^{(k)} + b^{(k)}) \\ Y^p &= g(W^{(L)}a^{(L)} + b^{(L+1)}). \end{aligned} \tag{3.30}$$

Unlike the forward propagation of the FNN and ResNet in section 3.2.1 and 3.2.2 there is an extra term with a weight matrix $V^{(k)}$ and an input value $x^{(k)}$ for each hidden layer.

More sophisticated forward propagation algorithms have later been proposed. Several of them arise from the ResNet forward propagation and are described by the following set

of equations

$$\begin{aligned} a^{(k)} &= a^{(k-1)} + \Delta t \sigma(W^{(k-1)} a^{(k-1)} + V^{(k)} x^{(k)} + b^{(k)}) \\ Y^p &= g(W^{(L)} a^{(L)} + b^{(L+1)}). \end{aligned} \tag{3.31}$$

In this thesis this method will be referred to as *Standard RNN*

The number states being fed to the network depends on the depth of the network. A network with L hidden layers will be able to consider the L previous states in time. The length of the memory of the network is therefore considered to be L.

When a RNN is predicting on sequential data, such as a riser simulation, the procedure is as follows: The state L time-steps before the current state will be fed from the first input layer to the first hidden layer of the RNN. This information will be treated by the first hidden layer and then passed on to the second hidden layer. The second hidden layer therefore receives information from the first hidden layer as well as the next state from the second input layer. The information is then treated by the second hidden layer before it is propagated forward. The same procedure is executed for the following layers until the information reaches the last layer where the prediction is made.

The main issue with RNNs is the familiar problem of the vanishing/exploding gradient and there has been made new architectures based on the original idea. Such architectures are known as *long short term memory* (LSTM) and *gated recurrent units* (GRU). These methods have mechanisms to let the network forget old data and focus on what it believes to be important [HS97].

Dynamical system approach

The same analogy described in section 3.2.2 between ResNet and dynamical systems can be drawn for RNNs. This was done in [CCHC19]. The theory is almost exactly the same and an anti-symmetric and Hamiltonian approach can be applied to make new and more stable forms of forward propagation. The anti-symmetric forward propagation is defined as

$$\begin{aligned} a^{(k)} &= a^{(k-1)} + \Delta t \sigma(K^{(k-1)} a^{(k-1)} + V^{(k)} x^{(k)} + b^{(k)}) \\ Y^p &= g(W^{(L)} a^{(L)} + b^{(L+1)}) \end{aligned}$$

where $K^{(k-1)} = C^{(k-1)} - \left(C^{(k-1)}\right)^T - \gamma I$ and all the other variables are as before. The term γI is added to include some diffusion in the system. This forward propagation defines the architectures hereby referred to in this thesis as *Anti-Symmetric RNNs*.

The forward propagation based on the Hamiltonian approach described in 3.2.2, which defines the architecture hereby is referred to as *Hamiltonian RNN*, is given by

$$\begin{aligned} a^{(k)} &= \begin{cases} 2a^{(k-1)} + \Delta t^2 \sigma(K^{(k-1)} a^{(k-1)} + V^{(k)} x^{(k)} + b^{(k)}), & k = 1 \\ 2a^{(k-1)} - a^{(k-2)} + \Delta t^2 \sigma(K^{(k-1)} a^{(k-1)} + V^{(k)} x^{(k)} + b^{(k)}), & \text{otherwise} \end{cases} \\ Y^p &= g(W^{(L)} a^{(L)} + b^{(L+1)}), \end{aligned}$$

where $K^{(k)} = -\left(C^{(k-1)}\right)^T C^{(k-1)}$.

Backpropagation through time

As previously mentioned in section 3.2.1 one may use a gradient based optimization method to train an ANN. To do so the gradient needs to be calculated. This is done by propagating backwards from the value of the error function and to the first input value. The method of finding the gradient of a RNN is known as backpropagation through time (BPTT). Using the same error function as in (3.19) one can derive a formula for the gradient of the Standard RNN.

Through the derivations of the gradients superscript indicating which layer the value belongs to will no longer include parentheses. This is to ease notation.

$$\begin{aligned}
\frac{\partial E}{\partial W_{i,j}} &= \frac{1}{2}\sum_{l=1}^{m}\frac{||Y_l^p - Y_l^t||_2^2}{\partial W_{i,j}} = \sum_{l=1}^{m}(Y_p - Y_T)_l^T\frac{\partial Y_p^l}{\partial W_{i,j}} \\
&= \sum_{l=1}^{m}(Y_p - Y_T)_l^T\frac{\partial}{\partial W_{i,j}}g(Wa^L + b^L) = \sum_{l=1}^{m}(Y_p - Y_T)_l^T g'(z^{L+1})e_i e_j^T a_l^L \\
&= \sum_{l=1}^{m}(Y_p - Y_T)_{i,l}g'(z_{i,l}^{L+1})a_{j,l}^L = \sum_{l=1}^{m}((Y_p - Y_T)\odot g'(z^{L+1}))_{i,l}(a^L)_{l,j}^T \\
&= \left(\left((Y_p - Y_T)\odot g'(z^{L+1})\right)(a^L)^T\right)_{i,j}.
\end{aligned}$$

And so we get that

$$\frac{\partial E}{\partial W} = \left((Y_p - Y_T)\odot g'(z^{L+1})\right)(a^L)^T.$$

By using the chain rule we are also able to evaluate the derivatives of the error function with respect to the parameters in the other matrices as well.

$$\begin{aligned}
\frac{\partial E}{\partial V_{i,j}^L} &= \frac{1}{2}\sum_{l=1}^{m}\frac{||Y_l^p - Y_l^t||_2^2}{\partial V_{i,j}^L} = \sum_{l=1}^{m}(Y_p - Y_T)_l^T\frac{\partial Y_p^l}{\partial V_{i,j}^L} \\
&= \sum_{l=1}^{m}(Y_p - Y_T)_l^T\frac{\partial}{\partial V_{i,j}^L}g(Wa^L + b^L) = \sum_{l=1}^{m}(Y_p - Y_T)_l^T g'(z^{L+1})W\frac{\partial a_l^L}{\partial V_{i,j}^L} \\
&= \sum_{l=1}^{m}(W^T((Y_p - Y_T)\odot g'(z^{L+1})))_l\frac{\partial a_l^L}{\partial V_{i,j}^L}.
\end{aligned}$$

The partial derivative $\frac{\partial a_l^L}{\partial V_{i,j}^L}$ can be found for the different forward propagation methods. In the case of the forward propagation defined in (3.31) we have that

$$\begin{aligned}
\frac{\partial a_l^L}{\partial V_{i,j}^L} &= \frac{\partial a_l^{L-1}}{\partial V_{i,j}^L} + \Delta t\sigma'(C^{L-1}a_l^{L-1} + V^L x_l^L)e_i e_j^T x_l^L \\
&= \Delta t\sigma'(C^{L-1}a_l^{L-1} + V^L x_l^L)e_i e_j^T x_l^L
\end{aligned}$$

If we let $\psi^L = W^T((Y_p - Y_T) \odot g'(z^{L+1}))$ and $z^L = C^{L-1}a_l^{L-1} + V^L x_l^L$ we may find the partial derivative of the cost function,

$$\begin{aligned}\frac{\partial E}{\partial V_{i,j}^L} &= \sum_{l=1}^{m} \psi_l^L \Delta t \sigma'(z_l^L) e_i e_j^T x_l^L = \Delta t \sum_{l=1}^{m} (\psi_l^L \odot \sigma'(z_l^L))_{i,l} x_{j,l}^L \\ &= \Delta t (\psi_l^L \odot \sigma'(z_l^L))(x^L)_{i,j}^T\end{aligned}$$

and we observe that

$$\frac{\partial E}{\partial V^L} = \Delta t \left(\psi_l^L \odot \sigma'(z_l^L)\right)(x^L)^T.$$

Calculating the derivative with respect to V^{L-1} is done in a similar fashion:

$$\begin{aligned}\frac{\partial E}{\partial V_{i,j}^{L-1}} &= \sum_{l=1}^{m} (W^T((Y_p - Y_T) \odot g'(z^{L+1})))_l^T \frac{\partial a_l^L}{\partial V_{i,j}^L} = \sum_{l=1}^{m} (\psi_l^L)^T \frac{\partial a_l^L}{\partial V_{i,j}^L} \\ \frac{\partial a_l^L}{\partial V_{i,j}^{L-1}} &= \frac{\partial a_l^{L-1}}{\partial V_{i,j}^{L-1}} + \Delta t \sigma'(z_l^L) C^{L-1} \frac{\partial a_l^{L-1}}{\partial V_{i,j}^{L-1}} \\ &= \left(1 + \Delta t \sigma'(z_l^L) C^{L-1}\right) \frac{\partial a_l^{L-1}}{\partial V_{i,j}^{L-1}} \\ \frac{\partial a_l^{L-1}}{\partial V_{i,j}^{L-1}} &= \Delta t \sigma'(C^{L-2} a_l^{L-2} + V^{L-1} x_l^{L-1}) e_i e_j^T x_l^{L-1} = \Delta t \sigma'(z_l^{L-1}) e_i e_j^T x_l^{L-1}\end{aligned}$$

using what we have found we can continue evaluating $\frac{\partial E}{\partial V_{i,j}^{L-1}}$.

$$\begin{aligned}\frac{\partial E}{\partial V_{i,j}^{L-1}} &= \sum_{l=1}^{m} (\psi_l^L) \frac{\partial a_l^L}{\partial V_{i,j}^L} = \sum_{l=1}^{m} \psi_l^L \left(1 + \Delta t \sigma'(z_l^L) C^{L-1}\right) \Delta t \sigma'(z_l^{L-1}) e_i e_j^T x_l^{L-1} \\ &= \sum_{l=1}^{m} \left(\psi^L + \Delta t (C^{L-1})^T (\psi^L \odot \sigma'(z^L))\right)_l \Delta t \sigma'(z_l^{L-1}) e_i e_j^T x_l^{L-1} \\ &= \sum_{l=1}^{m} \Delta t (\psi^{L-1} \odot \sigma(z^{L-1}))_l e_i e_j^T x_l^{L-1} = \sum_{l=1}^{m} \Delta t (\psi^{L-1} \odot \sigma(z^{L-1}))_{i,l} x_{j,l}^{L-1} \\ &= \Delta t ((\psi^{L-1} \odot \sigma(z^{L-1}))(x^{L-1})^T)_{i,j}\end{aligned}$$

and the matrix formulation of $\frac{\partial E}{\partial V^{L-1}}$ is given as

$$\frac{\partial E}{\partial V^{L-1}} = \Delta t ((\psi^{L-1} \odot \sigma(z^{L-1}))(x^{L-1})^T).$$

One may now see a pattern emerging where the next partial derivative may be found based on the previous ones. We may then write a more compact formula for finding the partial derivatives for V^{L-k}.

Let ψ^{L-k} be defined as

$$\psi^{L-k} = \begin{cases} W^T((Y_p - Y_T) \odot g'(z^{L+1}))), & \text{for } k = 0 \\ \psi^{L-k+1} + \Delta t \left(C^{L-k}\right)^T \left(\psi^{L-k+1} \odot \sigma'(z^{L-k+1})\right) & \text{otherwise} \end{cases}$$

then the partial derivatives of the error function are defined as

$$\begin{aligned}\frac{\partial E}{\partial V^{L-k}} &= \Delta t(\psi^{L-k} \odot \sigma(z^{L-k}))(x^{L-k})^T \\ \frac{\partial E}{\partial C^{L-k}} &= \Delta t(\psi^{L-k} \odot \sigma(z^{L-k}))(a^{L-k-1})^T \\ \frac{\partial E}{\partial b^{L-k}} &= \Delta t(\psi^{L-k} \odot \sigma(z^{L-k}))\mathbf{e}\end{aligned} \tag{3.32}$$

where $\mathbf{e}$ is a column vector of ones.

The BPTT algorithm for the other architectures can also be found by using the chain rule and does not differ much from the previous calculations. For the Vanilla RNN defined in (3.30) we have

$$\psi^{L-k} = \begin{cases} W^T((Y_p - Y_T) \odot g'(z^{L+1}))), & \text{for } k = 0 \\ \left(C^{L-k}\right)^T (\psi^{L-k+1} \odot \sigma'(z^{L-k+1}) & \text{otherwise,} \end{cases}$$

with partial derivatives given by (3.32). For the Anti-Symmetric RNN defined by Haber and Ruthotto [HR17, CCHC19] we have

$$\psi^{L-k} = \begin{cases} (W^T((Y_p - Y_T) \odot g'(z^{L+1}))), & \text{for } k = 0 \\ \psi^{L-k+1} + \Delta t(K^{L-k})^T(\psi^{L-k+1} \odot \sigma'(z^{L-k+1}), & \text{otherwise} \end{cases}$$

where $K^{L-k} = C^{L-k} - (C^{L-k})^T - \gamma I$. The partial derivatives are defined as

$$\begin{aligned}\frac{\partial E}{\partial V^{L-k}} &= \Delta t(\psi^{L-k} \odot \sigma(z^{L-k}))(x^{L-k})^T \\ \frac{\partial E}{\partial C^{L-k}} &= \Delta t(\psi^{L-k} \odot \sigma(z^{L-k}))(a^{L-k-1})^T - a^{L-k-1}(\psi^{L-k} \odot \sigma(z^{L-k}))^T \\ \frac{\partial E}{\partial b^{L-k}} &= \Delta t(\psi^{L-k} \odot \sigma(z^{L-k}))\mathbf{e}.\end{aligned} \tag{3.33}$$

For the Hamiltonian RNN based on [HR17] we have

$$\psi^{L-k} = \begin{cases} (W^T((Y_p - Y_T) \odot g'(z^{L+1}))), & \text{for } k = 0 \\ 2\psi^L + \Delta t^2 K^{L-1}(\psi^L \odot \sigma'(z^L)), & \text{for } k = 1 \\ 2\psi^{L-k+1} + \Delta t^2 K^{L-k}(\psi^{L-k+1} \odot \sigma'(z^{L-k+1})) - \psi^{L-k+2}, & \text{otherwise} \end{cases}$$

where $K^{L-k} = -(C^{L-k})^T C^{L-k}$. Using ψ we can write the partial derivatives as

$$\begin{aligned}\frac{\partial E}{\partial V^{L-k}} &= \Delta t^2(\psi^{L-k} \odot \sigma(z^{L-k}))(x^{L-k})^T \\ \frac{\partial E}{\partial C^{L-k}} &= -\Delta t^2 C^{L-k}((\psi^{L-k+1} \odot \sigma(z^{L-k+1}))(a^{L-k})^T + a^{L-k}(\psi^{L-k+1} \odot \sigma(z^{L-k+1}))^T) \\ \frac{\partial E}{\partial b^{L-k}} &= \Delta t^2(\psi^{L-k} \odot \sigma(z^{L-k}))\mathbf{e}.\end{aligned} \tag{3.34}$$

Normal equation

The minimization of the error function is by definition a non-linear least squares problem and may be written as

$$E = \frac{1}{2}\sum_{l=1}^{m} ||Y_l^p - Y_l^t||_2^2 = \frac{1}{2}\sum_{l=1}^{m} r_l^2 = \frac{1}{2} r^T r. \tag{3.35}$$

To find a minimizer to E with a gradient based optimization algorithm the gradient of E, given by

$$\nabla E = \sum_{l=1}^{m} (\nabla r_l) r_l = Jr,$$

can be utilized. Here r is the residual vector and J is the Jacobian of the residual vector. One may also find the Hessian of E,

$$\nabla^2 E = J^T J + \sum_{l=1}^{m} (\nabla^2 r_l) r_l,$$

which may be used in a second order optimization algorithm.

Since part of the Hessian contains the Jacobian we can use this information to find a better search direction than just the negative gradient [NW06]. To do so we will need to find the gradient of each of the residuals r_l. This is however quite easily derived from the earlier calculations of the gradient of E:

$$\begin{aligned}
\frac{\partial r_l}{\partial W} &= \frac{\partial ||Y_l^p - y_T^l||_2}{\partial W} = \frac{(Y_l^p - y_T^l)^T}{||Y_l^p - y_T^l||_2} g'(W a_l^L + b_w) a_l^L \\
&= \frac{(Y_l^p - y_T^l) \odot g'(W a_l^L + b_w)}{||Y_l^p - y_T^l||_2} a_l^L = \delta_l^L a_l^L
\end{aligned}$$

if we, for the forward propagation of the Standard RNN defined in (3.31), again define auxiliary variable ψ as

$$\psi_l^L = \begin{cases} W^T \delta_l^L, & \text{for } k = 0 \\ \psi_l^{L-k+1} + \Delta t (C^{L-k})^T (\psi_l^{L-k+1} \odot \sigma'(z_l^{L-k+1})), & \text{otherwise} \end{cases}$$

we can without much extra effort calculate the partial derivative of each residual r_l with respect to the weight matrices.

$$\begin{aligned}
\frac{\partial r_l}{\partial C^{L-k}} &= \Delta t (\psi_l^{L-k+1} \odot \sigma(z_l^{L-k+1}))(a_l^{L-k})^T \\
\frac{\partial r_l}{\partial V^{L-k}} &= \Delta t (\psi_l^{L-k} \odot \sigma(z_l^{L-k}))(x_l^{L-k})^T \\
\frac{\partial E}{\partial b^{L-k}} &= \Delta t \psi_l^{L-k} \odot \sigma(z_l^{L-k})
\end{aligned}$$

The same procedure follows for the different types of architectures.

The Anti-Symmetric architecture has partial derivatives of the residuals being

$$\begin{aligned}
\frac{\partial r_l}{\partial C^{L-k}} &= \Delta t(\psi_l^{L-k+1} \odot \sigma(z_l^{L-k+1}))(a_l^{L-k})^T \\
\frac{\partial r_l}{\partial V^{L-k}} &= \Delta t(\psi_l^{L-k} \odot \sigma(z_l^{L-k}))(a_l^{L-k-1})^T - a_l^{L-k-1}(\psi_l^{L-k} \odot \sigma(z_l^{L-k}))^T \\
\frac{\partial E}{\partial b^{L-k}} &= \Delta t \psi_l^{L-k} \odot \sigma(z_l^{L-k})
\end{aligned}$$

and the Hamiltonian approach gives

$$\begin{aligned}
\frac{\partial r_l}{\partial C^{L-k}} &= \Delta t^2(\psi_l^{L-k+1} \odot \sigma(z_l^{L-k+1}))(a_l^{L-k})^T \\
\frac{\partial r_l}{\partial V^{L-k}} &= -\Delta t^2 C^{L-k}((\psi_l^{L-k+1} \odot \sigma(z_l^{L-k+1}))(a_l^{L-k})^T + a_l^{L-k}(\psi_l^{L-k+1} \odot \sigma(z_l^{L-k+1}))^T) \\
\frac{\partial E}{\partial b^{L-k}} &= \Delta t^2 \psi_l^{L-k} \odot \sigma(z_l^{L-k})
\end{aligned}$$

Regularization

Regularizers are used in ANNs to reduce overfitting. This is done by adding certain constraints and/or penalties to the parameters.

The regularization technique known as *weight decay* is possibly the most common regularizer in the deep learning community [HR17]. This method penalizes large weights by setting the regularization function to be

$$R(\mathbf{K}) = \frac{1}{2}\sum_j ||K^j||_F^2 \tag{3.36}$$

where we denote the Frobenius norm by $||\cdot||_F$. This should increase the ability of the network to generalize because it reduces the chance of overfitting [ZF13, SHK+14].

This type of regularizer reduces the size of the weights, but does not affect the change in the weights between each layer. Given the interpretation of the forward propagation being a discretization of a dynamical system indicates that the weight matrix should be smooth or piecewise smooth in time. For better stability properties the authors of [HR17] propose using a regularizer on the form

$$R(\mathbf{K}) = \frac{\alpha}{2}\sum_j ||K^j - K^{j-1}||_F^2, \tag{3.37}$$

where α can be any positive constant, they use $\alpha = (\Delta t)^{-1}$. For the Standard RNN we can impose the regularizer on $\mathbf{C}$, $\mathbf{V}$ and $\mathbf{b}$. We then get

$$R(\mathbf{C},\mathbf{V},\mathbf{b}) = \frac{\alpha}{2}\left(\sum_{j=1}^{L}||V^j - V^{j-1}||_F^2 + \sum_{j=1}^{L-1}||C^j - C^{j-1}||_F^2 + \sum_{j=1}^{L}||b^j - b^{j-1}||_F^2\right).$$

For the weight matrix W between the last hidden layer and the output layer we propose using just the Frobenius norm to ensure that the weights are kept reasonably small.

Adding a regularizer will not increase workload noteworthy when finding the gradient of the new cost function

$$\hat{E} = E + R(\mathbf{C}, \mathbf{V}, \mathbf{b}) + \frac{1}{2}||W||_F^2.$$

For W we get that

$$\frac{\partial \hat{E}}{W} = \frac{\partial E}{W} + W$$

and for $\mathbf{V}$, $\mathbf{b}$ and $\mathbf{C}$ in the Standard RNN we have

$$\frac{\partial \hat{E}}{V^{L-k}} = \frac{\partial E}{V^{L-k}} + \alpha \begin{cases} V^L - V^{L-1} & \text{for } k = 0 \\ -V^{L-k+1} + 2V^{L-k} - V^{L-k-1} & \text{for } 0 < k < L \\ -(V^1 - V^0) & \text{for } k = L \end{cases}$$

$$\frac{\partial \hat{E}}{b^{L-k}} = \frac{\partial E}{b^{L-k}} + \alpha \begin{cases} b^L - b^{L-1} & \text{for } k = 0 \\ -b^{L-k+1} + 2b^{L-k} - b^{L-k-1} & \text{for } 0 < k < L \\ -(b^1 - b^0) & \text{for } k = L \end{cases}$$

$$\frac{\partial \hat{E}}{C^{L-k}} = \frac{\partial E}{C^{L-k}} + \alpha \begin{cases} C^{L-1} - C^{L-2} & \text{for } k = 1 \\ -C^{L-k+1} + 2C^{L-k} - C^{L-k-1} & \text{for } 1 < k < L \\ -(C^1 - C^0) & \text{for } k = L \end{cases}.$$

For the dynamical system approaches we have restrictions on C meaning the partial derivatives with respect to C will be different. For the Anti-Symmetric RNNs we have

$$\frac{\partial \hat{E}}{C^{L-k}} = \frac{\partial E}{C^{L-k}} + 2\alpha \begin{cases} K^{L-1} - K^{L-2} & \text{for } k = 1 \\ -K^{L-k+1} + 2K^{L-k} - K^{L-k-1} & \text{for } 1 < k < L \\ -(K^1 - K^0) & \text{for } k = L \end{cases}.$$

with $K^j = C^j - (C^j)^T$. For the Hamiltonian approach we have

$$\frac{\partial \hat{E}}{C^{L-k}} = \frac{\partial E}{C^{L-k}} + \alpha C^{L-k} \begin{cases} K^{L-1} - K^{L-2} & \text{for } k = 1 \\ -K^{L-k+1} + 2K^{L-k} - K^{L-k-1} & \text{for } 1 < k < L \\ -(K^1 - K^0) & \text{for } k = L \end{cases}$$

where $K^j = (C^j)^T C^j$.

3.3 Optimization

In the previous sections we showed how to evaluate the gradient for the different architectures. The gradient is to be used in optimization algorithms. The choice of optimization

algorithm has an impact on how fast, but also on how well the system is trained once the algorithm converges [HR17]. In this section information about the optimization algorithms used in the experiments will be presented.

The number of parameters in a RNN with L hidden layers is given by

$$N = n_1^2(L-1) + n_1 n_2 L + n_3 n_1 + L n_1 + n_3.$$

This is due to the different dimensions of the weight matrices in the RNN. The matrix C^k is an $n_1 \times n_1$ matrix, V^k is an $n_1 \times n_2$ matrix and W is an $n_3 \times n_1$ matrix. The biases are vectors of size n_1 in the hidden layers and n_3 in the output layer.

Observe that the gradient of the cost function with respect to all the data points can be calculated in $O(Nm)$ flops, where m is the number of data points. The gradient can however be evaluated by calculating the contribution to the gradient from each data point in parallel. Additionally, the matrix multiplication in the derivations above can also be calculated in parallel implying that the gradient can be found very fast if enough processors are available.

3.3.1 Gradient descent

There are different algorithms with different properties for optimization. The gradient descent method is maybe the simplest to implement and each iteration can be very fast. The gradient descent algorithm can be found in algorithm 1

Data: Gradient descent(x_0, f, ∇f, ρ, c_1)
Result: Returns a minimizer x^* of $f(x)$
$x = x_0$
while *Not converged* **do**
  $p = -\nabla f(x)$
  $\alpha = 1$
  while $f(x + \alpha \nabla f^T p) \geq f(x) + c_1 \alpha \nabla f(x)^T p$ **do**
    $\alpha = \rho\alpha$
  end
  $x = x + \alpha p$
end

Algorithm 1: Gradient descent

where the inequality condition in the inner while loop is known as the sufficient decrease condition or perhaps more famously as the Armijo condition. The constant c_1 is often chosen to be small e.g. 10^{-3}.

To get an upper bound on the convergence rate of the gradient descent method we consider the ideal case of minimizing the quadratic function

$$f(x) = \frac{1}{2}x^T Q x - b^T x$$

where Q is a symmetric positive definite. We then know that the solution is given as $x^* = Q^{-1}b$. We also assume we are using exact line search given by $\alpha = \frac{||\nabla f||_2^2}{||\nabla f||_Q^2}$.

One can then show that

$$\begin{aligned}||x_{k+1} - x^*||_Q^2 &= \left(1 - \frac{||\nabla f||_2^2}{||\nabla f||_Q^2 ||\nabla f||_{Q^{-1}}^2}\right) ||x_k - x^*||_Q^2 \\ &\leq \left(\frac{\kappa(Q) - 1}{\kappa(Q) + 1}\right) ||x_k - x^*||_Q^2,\end{aligned}$$

see [NW06], and therefore we have linear convergence. It is also worth noticing that if Q is poorly conditioned, then the convergence rate will be very slow.

Given our non-linear optimization problem we cannot expect any better convergence than for this ideal case. If the Armijo condition are used instead of exact line search then the order of convergence is the same, but one can expect a larger constant, resulting in more iterations [NW06]. This will make the training time extremely large and the method might therefore not be the optimal choice for training ANNs since we are able to extract more information about the problem for a little more effort.

The computational cost of the gradient descent in each iteration is given by the cost of calculating gradient of the data points, calculating the forward propagation in the line search and updating the parameters, giving a total cost of $O(Nm)$ flops. All these calculations are however parallelizable and can be calculated very fast.

3.3.2 Newton method

It would be possible to evaluate the Hessian of the architecture and use it for a Newton method. Newton methods have quadratic convergence once the iterates are close enough to a minimizer. This is an attractive property, but it comes at the cost of solving a linear system of size $N \times N$ where N is the number of parameters in system. The number of parameters will be very large for deep artificial neural networks and the cost for solving the dense Hessian would be $O(N^3)$ flops per iteration, which is not be feasible. Additionally, the cost of evaluating the Hessian for the neural network is large and so other optimization methods not including the Hessian are usually preferred.

3.3.3 Quasi-Newton method

The next possibility would the be to use a quasi-Newton method. This class of method was first introduced by W.C. Davidon in 1959 [Dav91], but was only made famous when a paper written by R. Fletcher and M.J.D. Powell was published in 1963 [FP63, GW18]. The different quasi-Newton methods store an approximation of the Hessian matrix based on the previous calculated gradients and update this matrix for each iteration. We are therefore relieved from the task of evaluating the Hessian at every iteration. A sophisticated implementation also allows for a fast solution of the linear system which must be solved to calculate the search direction.

The quasi-Newton method has super-linear convergence once the iterates get close enough to the solution and is commonly used in many software packages. One of the most common method is the BFGS-method invented by Charles George Broyden, Roger

Fletcher, Donald Goldfarb and David Shanno individually in 1970 [NW06, GW18]. The BFGS algorithm is given in algorithm 2.

Data: BFGS Quasi-Newton(x_0, f, ∇f, ρ, c_1)
Result: Returns a minimizer x^* of $f(x)$
$x = x_0$
$B_0 = I$
$k = 0$
while *Not converged* **do**
  $p_k = B_k^{-1}(-\nabla f(x_k))$
  $\alpha = 1$
  while $f(x_k + \alpha \nabla f(x_k)^T p_k) \geq f(x_k) + c_1 \alpha \nabla f(x_k)^T p_k$ **do**
    $\alpha = \rho\alpha$
  end
  $x_{k+1} = x_k + \alpha p_k$
  $d_k = \alpha p_k$
  $y_k = \nabla f(x_k + \alpha p_k) - \nabla f(x_k)$
  $B_{k+1} = B_k - \frac{B_k d_k d_k^T B_k}{d_k^T B_k d_k} + \frac{y_k y_k^T}{y_k^T d_k}$
  $k = k + 1$
end

Algorithm 2: Quasi-Newton (BFGS)

In the given algorithm there is no need to evaluate the Hessian matrix. The Hessian approximation, B_k, is positive definite if

$$y_k^T d_k > 0 \tag{3.38}$$

[NW06]. Since we do not want an indefinite approximation to the Hessian at the solution we can evaluate (3.38) to decide whether to accept the update or not.

Since we know the matrix B_k is symmetric positive definite we can use familiar fast converging algorithms such as a preconditioned conjugate gradient to solve the linear system that arises.

One can however use the information we know about the matrix always being positive definite and symmetric to improve the method [GW18]. Instead of updating the actual matrix B_k one can store the Cholesky decomposition $R_k^T R_k$ and rather update upper triangular matrix R_k at each iteration. This way we need to solve two triangular linear systems each taking $O(N^2)$ flops, reducing the computational cost of an order of magnitude.

To ensure that each iteration has a complexity of $O\left(N^2\right)$ we need to update the Cholesky decomposition with the same complexity or lower. A method for doing so was proposed in [GM72] and is described [GW18] and in what follows

The update B_{k+1} defined in line 15 in algorithm 2 given by

$$B_{k+1} = B_k - \frac{B_k d_k d_k^T B_k^T}{d_k^T B_k d_k} + \frac{y_k y_k^T}{y_k^T d_k}$$

can be written on the form

$$\begin{aligned} B_{k+1} &= (I - v_k d_k^T) B_k (I - d_k v_k^T) \\ &= (I - v_k d_k^T) R_k^T R_k (I - d_k v_k^T) \\ &= \hat{R}_k^T \hat{R}_k \end{aligned}$$

with $\hat{R}_k := R_k(I - d_k v_k^T)$. We have used that $B_k = R_k^T R_k$ and

$$v_k = \frac{B_k d_k}{d_k^T B_k d_k} \pm \frac{y_k}{(y_k^T d_k)^{1/2} (d_k^T B_k d_k)^{1/2}}.$$

After some algebra we get that

$$\hat{R}_k = R_k + \frac{R_k d_k}{||R_k d_k||} \left(\pm \frac{y}{(y_k^T d_k)^{1/2}} - \frac{R_k^T R_k d_k}{||R_k d_k||} \right)^T. \tag{3.39}$$

By defining

$$w = \frac{R_k d_k}{||R_k d_k||} \quad \text{and} \quad z = \pm \frac{y}{(y_k^T d_k)^{1/2}} - R_k^T w$$

we get a shorter expression for (3.39)

$$\hat{R}_k = R_k + w z^T.$$

Note that $||w||_2 = 1$.

Next we use Givens rotations to transform $\hat{R}_k$ into upper triangular form, and get an upper triangular matrix $R_{k+1} = Q\hat{R}_k$ such that

$$B_{k+1} = \hat{R}_k^T \hat{R}_k = \hat{R}_k^T Q^T Q \hat{R}_k = R_{k+1}^T R_{k+1}.$$

which is the desired updated Cholesky factorization of B_{k+1}.

This can first be done by applying a set of Givens rotations Q_1 such that

$$Q_1 w = e_n$$

where e_n is a vector with 1 as its last element and zero otherwise. When applied to $\hat{R}_k$ we get

$$Q_1 \hat{R}_k = Q_1 R_k + e_n z^T$$

which is an upper triangular matrix except for the last row which is non-zero after the rotations. Applying a second set of Givens rotations Q_2 to rotate the matrix back to an upper triangular matrix gives the desired outcome. And so we have

$$R_{k+1} = Q_2 Q_1 \hat{R}_k.$$

Each Givens rotation uses $O(N)$ flops and the rotations are applied $O(N)$ times. Consequently the method uses $O(N^2)$ to calculate the new Cholesky decomposition at every iteration. Solving a triangular system can be done in $O(N^2)$ operations and so we have a method where each iteration is one order of magnitude faster than a classical Newton method and standard BFGS. Still it has the exact same convergence properties as the BFGS if we assume infinite precision. How the improved BFGS methods compares to the gradient descent and other optimization methods will be tested in chapter 4.

Selecting optimal sign in z

Since the first term in z is valid for both a positive or a negative sign means that there are two possible updates for the Cholesky decomposition. The update that gives the most stable algorithm is the updated with the smallest value [GW18]. Using the properties of the Euclidean norm for a rank-one matrix and the fact that $||w||_2 = 1$ then

$$||wz^T||_2 = ||w||_2||z||_2 = ||z||_2.$$

This implies that the sign for the first term in z should be chosen such that z is minimized.

The Euclidean norm of z is given by

$$\left|\left|\pm\frac{y}{\sqrt{y^T d}} - \frac{Bd}{\sqrt{d^T Bd}}\right|\right|_2^2 = \frac{y^T y}{y^T d} \mp 2\frac{y^T Bd}{\sqrt{y^T d}\sqrt{d^T Bd}} + \frac{d^T B^2 d}{d^T Bd}. \tag{3.40}$$

using that $Bd/\alpha = -g$ we get the following way of choosing the sign.

$$z = -R_k^T w + \frac{y}{\sqrt{y^T d}} \begin{cases} 1 & \text{if } y^T g < 0 \\ -1 & \text{otherwise.} \end{cases} \tag{3.41}$$

See [GW18].

The main computational difference between the gradient descent and the BFGS method with Cholesky decomposition is the cost of solving the system of equations. This BFGS method will therefore have a cost of $O(Nm + N^2)$ flops per iteration. The calculation of the gradient can easily be parallelized while the triangular system of equations can not as easily be solved in parallel.

3.3.4 Gauss-Newton method

A method also proven to be effective is the Gauss-Newton method [HR17]. This method is advantageous since we wish to minimize the cost function (3.19) which then is a non-linear least squares problem on the form

$$\min_{W^k, b^k} \frac{1}{2} \sum_{l=1}^{m} ||Y_l^p - Y_l^t||_2^2 \tag{3.42}$$

by definition. The objective function in (3.42) can be rewritten as

$$E = \frac{1}{2} \sum_{l=1}^{m} r_i^2 = \frac{1}{2} r^T r, \qquad \text{with } r_i = ||Y_i^p - Y_i^t||_2.$$

We then have the following relations when calculating the gradient, ∇E, and Hessian, $\nabla^2 E$, of the cost function E with respect to the weight and biases

$$\nabla E = \sum_{l=1}^{m} r_l(\nabla r_l) = J^T r$$
$$\nabla^2 E = \sum_{l=1}^{m} \nabla r_l \nabla r_l^T + \sum_{l=1}^{m} r_l(\nabla^2 r_l) = J^T J + \sum_{l=1}^{m} r_l(\nabla^2 r_l).$$

Here J is the Jacobian matrix of the residual vector r with the lth row given by ∇r_l^T. If, when the iterates are approaching a minimizer, the $J^T J$ term dominates the $\sum_{l=1}^{m} r_l(\nabla^2 r_l)$ term we can expect similar convergence to Newton's method[NW06] when using the search direction p found by solving

$$J^T J p = -J^T r \tag{3.43}$$

in the Gauss-Newton method. The Gauss-Newton method can also be used to exploit the size of the system. This is because the search direction p found in the $N \times N$ system (3.43) is the same as the solution to

$$\min_p ||Jp + r||_2.$$

This method will work best for small training sets as we then will have a Jacobian of the rank no greater than the size of the training set. This system can then be solved using a version of the conjugate gradient method for normal equations known as CGNR [Saa03]. By using such a method there will be no need for evaluating the expensive matrix product $J^T J$.

This optimization method is known as the Gauss-Newton method and the algorithm is given by algorithm 3 [NW06]. The search direction is calculated using the conjugate gradient method for normal equations defined in algorithm 4.

Data: Gauss-Newton(x_0, r, J)
Result: Returns a minimizer x^* of $f(x)$
$x = x_0$
while *Not converged* **do**
  $p = \min_p ||J(x)p + r(x)||_2$
  $\alpha = 1$
  while $f(x + \alpha r(x)^T J(x)p) \geq f(x) + c_1 \alpha r(x)^T J(x)p$ **do**
    $\alpha = \rho\alpha$
  end
  $x = x + \alpha p$
end

Algorithm 3: Gauss Newton

Method such as the CGNR may have poor convergence properties as it is attempting to solve the following problem

$$J^T J x = J^T r \tag{3.44}$$

where the condition number of $J^T J$ can be very large. This will affect the convergence which is given by

$$||x^* - x_n||_{J^T J} \leq 2 \frac{\sqrt{\kappa(J^T J)} - 1}{\sqrt{\kappa(J^T J)} + 1} ||x^* - x_0||_{J^T J} \tag{3.45}$$

where $\kappa(J^T J)$ is the condition number of $J^T J$. Equation (3.45) can also be expressed using the singular values of J

$$||x^* - x_n||_{J^T J} \leq 2 \frac{\sigma_N - \sigma_0}{\sigma_N + \sigma_0} ||x^* - x_0||_{J^T J} \tag{3.46}$$

where σ_N and σ_1 is the largest and smallest singular value of J [Saa03].

Data: CGNR(b, J, $x0$)
Result: Returns a solution to $J^TJx = Jb$
$r_0 = b - Jx_0$, $z_0 = J^Tr_0, p_0 = z_0$
while *not converged* **do**
$w_i = Jp_i$
$\alpha_i = \frac{||z_i||^2}{||w_i||^2}$
$x_{i+1} = x_i + \alpha_i p_i$
$r_{i+1} = r_i - \alpha_i w_i$
$z_{i+1} = J^T r_{i+1}$
$\beta_i = \frac{||z_{i+1}||^2}{||z_i||^2}$
$p_{i+1} = z_{i+1} + \beta_i p_i$
end
return x

Algorithm 4: CGNR

An implementation of the Gauss-Newton method using the CGNR algorithm for solving the linear system of equations will use $O(Nm \min(N, m))$ flops in each iteration. The CGNR algorithm will converge within $\min(N, m)$ iterations as this is the maximum number of distinct eigenvalues for the matrix J^TJ. This means that the linear system can be solved in $O(Nm \min(N, m))$ flops. The cost of finding the gradient is an order of magnitude smaller and is therefore not included in the big-O notation.

3.3.5 L-BFGS

The L-BFGS method is a method that was created for problems with a large set of parameters such that the Hessian approximation was not possible to store or invert. The method is based on writing the update of the inverse Hessian matrix M_{k+1} as

$$M_{k+1} = \left(I - \frac{1}{d_k^T y_k} d_k y_k^T\right) M_k \left(I - \frac{1}{d_k^T y_k} y_k d_k^T\right) + \frac{1}{d_k^T y_k} d_k d_k^T \tag{3.47}$$

[Gil18]. We may write (3.47) as

$$M_{k+1} = V_k^T M_k V_k + \frac{1}{d_k^T y_k} d_k d_k^T, \qquad \text{where } V_k = I - \frac{1}{d_k^T y_k} y_k d_k^T. \tag{3.48}$$

Note that V_k is an oblique projection, see [Saa03].

Replacing M_k in (3.48) with

$$V_{k-1}^T M_{k-1} V_{k-1} + \frac{1}{d_{k-1}^T y_{k-1}} d_{k-1} d_{k-1}^T$$

we get that

$$M_{k+1} = V_k^T V_{k-1}^T M_{k-1} V_{k-1} V_k + \frac{1}{d_{k-1}^T y_{k-1}} V_k^T d_{k-1} d_{k-1}^T V_k + \frac{1}{d_k^T y_k} d_k d_k^T. \tag{3.49}$$

This implies that we can reconstruct the approximate inverse Hessian based on the k last iterations. Therefore, by storing the ν last vector pairs (y_k, d_k) one can calculate an approximation to the inverse Hessian without having to store the actual approximation M_k in the following way

$$M_{k+1} = \sum_{i=k}^{\nu+k+1} \left(\prod_{j=i}^{k+\nu-1} V_{i-\nu+1} \right)^T \frac{d_{i-\nu} d_{i-\nu}^T}{d_{i-\nu}^T y_{i-\nu}} \left(\prod_{j=i}^{k+\nu-1} V_{i-\nu+1} \right) + \\ + \frac{d_{k-1} d_{k-1}^T}{d_{k-1}^T y_{k-1}} + \left(\prod_{j=i}^{k+\nu-1} V_{i-\nu+1} \right)^T M_0 \left(\prod_{j=i}^{k+\nu-1} V_{i-\nu+1} \right).$$

However, if one is to keep the method from taking up too much memory then the matrices V_k should not be computed explicitly to find the next search direction. Rather we can use a recursive process to calculate the search direction

$$p_k = -M_k \nabla E$$

efficiently. The method is given in the following algorithm

Data: L-BFGS multiplication
$q = -\nabla E$
for $i = k-1, k-2, \ldots, k-\nu$ **do**
    $\sigma_i = \frac{d_i^T q}{d_i^T y_i}$
    $q = q - \sigma_i y_i$
end
$r = \frac{d_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}} q$
for $i = k-\nu, k-\nu+1, \ldots, k-1$ **do**
    $\beta = \frac{y_i^T r}{d_i^T y_i}$
    $r = r + d_i(\sigma_i - \beta)$
end

Algorithm 5: L-BFGS

To implement the L-BFGS method one simply has to store the ν last vector pairs and use algorithm 5 to find the search direction p_k in algorithm 2 instead of solving the full linear system. Algorithm 5 requires $4N\nu$ multiplications and so each iteration will therefore take $O(N\nu + Nm)$ flops where the term Nm comes from calculating the gradient, which is parallelizable.

3.3.6 Stochastic descent

Stochastic gradient descent is a very popular method in the machine learning context. It is very similar to the gradient descent method, but instead of calculating the gradient of the whole training set it only calculates the gradient of a single randomly drawn training sample [Bot10]. We therefore have the following possible updates for the learning problem

$$w_{k+1} = w_k - \gamma \nabla (\frac{1}{2} r_l^2), \tag{3.50}$$

where w_k is an array containing all the parameters of the ANN in the kth iteration, the index l is picked at random, r_l is the lth residual in the cost function and γ is known as the learning rate of the problem and is user defined.

This is however the simplest of the stochastic gradient descent methods. Today it is more common to calculate the gradient for a batch of data and then update the gradient accordingly. This method is known as mini-batch gradient descent and usually uses randomly drawn batches from the training set of sizes between 50 and 250 samples.

Stochastic gradient descent has proved to be efficient for large scale machine learning problems due to the fact that one does not need to use the whole training set for each iterations. Handling the whole training set is expensive when considering training time and memory, so even if each iteration is not as good as for instance a Newton step, it is able to perform many steps in a short amount of time. Additionally, due to the stochasticity of the method it is less likely to converge to a local minimum.

The cost of one iteration of a stochastic gradient descent method is $O(N\hat{m})$ flops, where $\hat{m}$ is the size of the mini-batches. Calculating the gradient based on the mini-batch can also be parallelized, just like the full gradient.

Adam

There has also been made improvements to the learning parameters and update methods to make the stochastic gradient descent method converge faster. Common algorithms in the machine learning environment today is the Adagrad algorithm [CDHS11] and the Adam algorithm [KB15]. These methods have adaptive choosing of the update for each iteration, making the method faster as they are less prone to noise in the gradient. The Adam method is considered to be the fastest among the current stochastic descent algorithms and the pseudo-code given in the original paper [KB15] is given in algorithm 6

Data: Adam(β_1, β_2, α, $E(\theta)$)
$m_0 = 0$
$v_0 = 0$
$t = 0$
$\epsilon = 10^{-8}$
while *not converged* **do**
  $t = t + 1$
  $g_t = \nabla f_t(\theta_{t-1})$
  $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  $\hat{m}_t = m_t / (1 - \beta_1^t)$
  $\hat{v}_t = v_t / (1 - \beta_2^t)$
  $\theta_t = \theta_t - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$
end
return θ_t

Algorithm 6: The Adam method as given in [KB15]

A method for improving the convergence speed of Adam was proposed in [Doz16]. Here they use a Nesterov momentum term in addition to the previously defined momentum term $\hat{m}_t$. This algorithm is known as Nadam and is equal to the Adam algorithm except

for the line defining $\hat{m}_t$ which will be set to

$$\hat{m}_t = \frac{m_t}{(1-\beta_1^t)} + \frac{1-\beta_1}{1-\beta_1^t} g_t.$$

Stochastic Gauss-Newton

Another stochastic optimization method is the stochastic version of the Gauss-Newton method [WRSN13]. This method works similarly to the previously described Gauss Newton method, but it calculates a Hessian approximation to the error function based on the Jacobian with respect to a mini-batch of data. Since the method of finding the Hessian approximation is parallelizable and if the size of the batches are small compared to the number of parameters in the ANN then finding a better search direction than the SGD can be done very efficiently.

We therefore have the following update for each iteration

$$w_{k+1} = w_k - \gamma(\hat{J}_k^T \hat{J}_k)^{-1} \hat{J}_k^T \hat{r}_k. \tag{3.51}$$

Here $\hat{J}_k$ is the Jacobian to the residual vector $\hat{r}_k$ which is based on a randomly drawn batch of data. The parameter γ is the learning rate as described in the introduction to this sub-section.

Each iteration in the stochastic Gauss-Newton method will have a cost of $O(N\hat{m}^2)$ flops, where $\hat{m}$ is the size of the mini-batch and is usually small compared to N. The matrix multiplications in the CGNR algorithm can also be parallelized making the method fast if enough processor units are available.

Chapter 4

Experiments

In the previous chapters the theory about the different artificial neural networks were presented together with brief underlying theory about riser mechanics. In this section the four artificial neural network architectures, *Anti-Symmetric*, *Hamiltonian*, *Standard* and *Vanilla RNN*, will be compared by their ability to generalize and handle noise. The machine learning models ability for accurately predicting properties of the risers bending moments will be evaluated. Additionally the different optimization methods will be considered.

4.1 Riser simulation

4.1.1 Initial setup

Due to the lack of a real training and test set a riser simulator was created based on the theory in section 3.1. This simulator was written using the Matlab programming language. For all the experiments in this chapter the same riser setup was used. The properties of the riser can be found in table 4.1.

L	EI	T	c	ρ
$100\,\mathrm{m}$	$3.186\times10^{5}\,\mathrm{Nm}^2$	$7.5537\times10^{6}\,\mathrm{N}$	2	$1200\,\mathrm{kg/m}^3$
C_d	C_I	R	A_e	ρ_w
2	2	$1/10\,\mathrm{m}$	$\pi/100\,\mathrm{m}^2$	$1000\,\mathrm{kg/m}^3$

Table 4.1: Parameters for the simulations

The forcing term in the system was calculated based on Morison's equation (3.4), Airy wave theory (3.5) and the different sea states found in table 4.2. The first three columns will be used when training the RNNs while the two last will be used for testing their ability to generalize.

	Training 1	Training 2	Training 3	Test 1	Test 2
Wave length (m)	76.5	136	212.2	106.25	174.1
Wave period (s)	8.6	11.4	14.3	10	12.85
Amplitude (m)	4.1	8.5	14.8	6.3	11.65

Table 4.2: The five different types of waves used in the training and test sets.

4.1.2 Spatial discretization

The governing riser equations were discretized with a finite element method in space. Since the information about the bending moments is of value, piecewise Hermite fifth order polynomials were chosen as test functions. This comes from the fact that the numerical solution can be written on the form

$$u_h(x,t) = \sum_{j=0}^{N} u_j(t)\phi_{3j}(x) + \theta_j(t)\phi_{3j+1}(x) + c_j(t)\phi_{3j+2}(x) \tag{4.1}$$

where $u_j(t)$, $\theta_j(t)$ and $c_j(t)$ are the value, the inclination and the curvature of the function at the point x_j. The Hermite function $\phi_{3j+k}(x)$ for $k = 0, 1, 2$ is 1 in its 0th, 1st or 2nd derivative and is 0 in the remaining two, at node j. This means we have three active basis functions in the jth node, hence the given subscript $3j + k$. The curvature can therefore easily be extracted from the solution and to later calculate bending moments.

The fifth order Hermite polynomials used is defined as

$$\begin{aligned}
\hat{\phi}_0(\hat{x}) &= -\frac{3}{16}x^5 + \frac{5}{8}x^3 - \frac{15}{16}x + \frac{1}{2} \\
\hat{\phi}_1(\hat{x}) &= -\frac{3h}{32}x^5 + \frac{h}{32}x^4 + \frac{5h}{16}x^3 - \frac{3h}{16}x^2 - \frac{7h}{32}x + \frac{5h}{32} \\
\hat{\phi}_2(\hat{x}) &= -\frac{h^2}{64}x^5 + \frac{h^2}{64}x^4 + \frac{h^2}{32}x^3 - \frac{h^2}{32}x^2 - \frac{h^2}{64}x + \frac{h^2}{64} \\
\hat{\phi}_3(\hat{x}) &= \frac{3}{16}x^5 - \frac{5}{8}x^3 + \frac{15}{16}x + \frac{1}{2} \\
\hat{\phi}_4(\hat{x}) &= -\frac{3h}{32}x^5 - \frac{h}{32}x^4 + \frac{5h}{16}x^3 + \frac{3h}{16}x^2 - \frac{7h}{32}x - \frac{5h}{32} \\
\hat{\phi}_5(\hat{x}) &= \frac{h^2}{64}x^5 + \frac{h^2}{64}x^4 - \frac{h^2}{32}x^3 - \frac{h^2}{32}x^2 + \frac{h^2}{64}x + \frac{h^2}{64}
\end{aligned} \tag{4.2}$$

where h is the length of the intervals in the discretization and $x \in [-1, 1]$. The polynomials are plotted in 4.1 with $h = 2$.

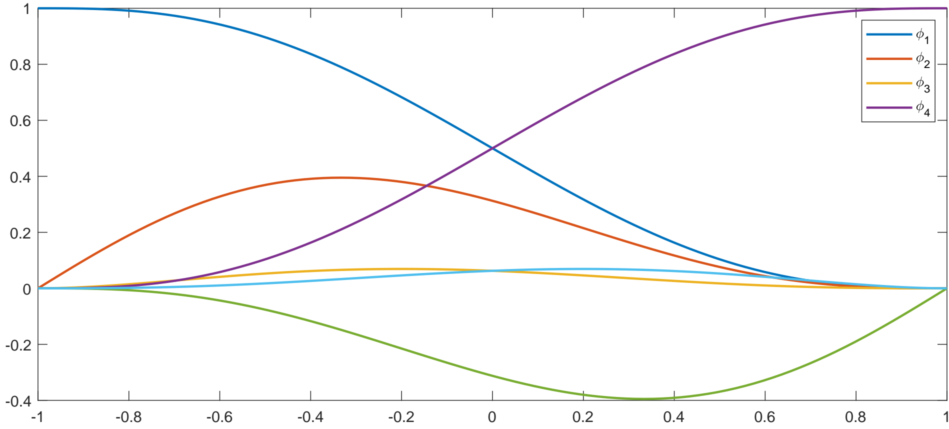

Figure 4.1: The Hermite shape functions on the standard interval $[-1, 1]$

Discretizing the 100 m long riser into 100 elements will give a sufficiently accurate solution in space. In this way the accuracy of the solution will not affect the overall goal of the experiments which is to evaluate the artificial neural networks ability to make predictions on the riser.

Discretizing the riser into 100 elements will result in a systems of ODEs given by equation (3.16) defined in section 3.1.3 of size 594×594. This system of ODEs is stiff.

4.1.3 Time integration

The stiff system of ODEs emerging from the spatial discretization was integrated using the Euler exponential integrator defined in (3.18). This first order exponential method worked well for the waves with properties given in table 4.2. The numerical solver ran into some instabilities due to the product

$$z^{-1}(e^{-z} - I) \tag{4.3}$$

which becomes hard to evaluate in finite precision when z becomes small. There exists stable methods for calculating (4.3), but they were not implemented for this thesis.

The step size in time was set to 0.01 as this gave stable simulations. Using this step size also had the advantage that it replicates the frequency at which the sensors send out information.

The forcing term on the riser was found by calculating the speed of the water particles within the waves using Airy wave theory. The maximum velocity profile for the different waves used in the training sets can be viewed in figure 4.2.

The simulations of the riser was run for 10000 time steps for the training and test sets. Information about the riser's displacement, inclination and curvature in every node was stored in separate comma separated value (CSV) files designed to be loaded by a script when training the artificial neural networks. Some snapshots of the riser's displacement in training set 1 can be examined in figure 4.3.

It is worth noting that due to the boundary conditions the riser does not move at either endpoints and so the distance from equilibrium close to the endpoints will also be small.

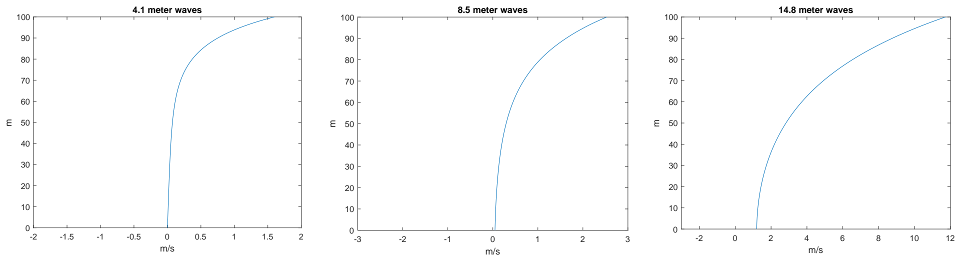

Figure 4.2: The blue line represents the current profile of the waves represented by Airy wave theory at maximum peak. Observe the different first axis for all the figures.

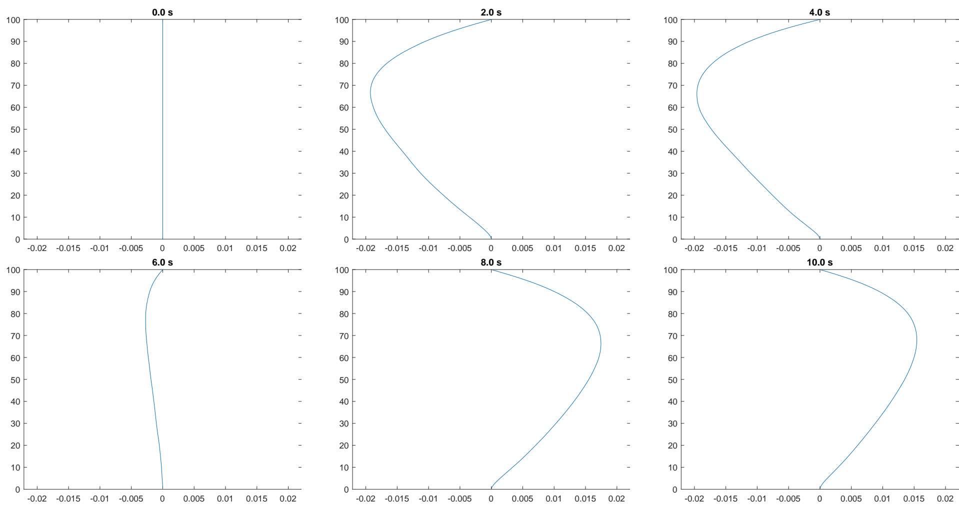

Figure 4.3: Adding waves and keeping the other parameters constant. The blue curve represents the riser. The waves correspond to the 8.5m waves in figure 4.2.

4.2 RNN Models

In this section the different RNNs introduced in chapter 3 will be compared to each other with respect to their ability to predict on the riser. The networks have different parameters which can be tuned. They all have the possibility of increasing the number of layers and also the size of each layer. They also have some specific parameters only applicable to them self.

All the RNNs except for the Vanilla RNN are based upon the assumption of the neural network being a dynamical system. This means that the step size Δt will be a hyper parameter we can vary to make the forward propagation more stable. Additionally, the Anti-Symmetric RNN also has a hyper parameter previously introduced as the diffusion constant γ which can be tuned. The effects these parameters have on the training time will be discussed as well as their corresponding RNNs abilities to generalize.

We will test and discuss how the number of sensors as well as their placement affect the machine learning models abilities to make accurate predictions. We also investigate the effect on the prediction performance due to adding different types of regularization to the ANN models. At last the different optimization methods are evaluated with respect to speed and convergence abilities for the RNNs.

Unless expressed otherwise the different networks were trained using the BFGS method with Cholesky decomposition described in section 3.3.3. All the experiments concerning the prediction abilities of the different neural network architectures were done using the training and tests set described in the previous section.

To evaluate the models the coefficient of determination, denoted R^2, is used. The R^2 values are given by the following formula

$$R^2 = 1 - \frac{\sum_i (f_i - y_i)^2}{\sum_i (\bar{y} - y_i)^2} \tag{4.4}$$

where f_i is the prediction on the ith sample in the data set, y_i is the corresponding true value and $\bar{y}$ is the mean of all the true values.

The R^2 values measures how well a regression model fit the data compared to a straight line through the data. Values close to 1 means the regression model is a good fit for the given data set, values less than 0 indicate that the model is a worse fit on the data than simply a straight line.

Features

In the following experiments the features are the information about the risers displacement. The main theme in all the experiments regarding the riser will be to make predictions based on data received from artificial sensors placed along the simulated riser. In practise this means extracting data from certain nodes in the numerical solution of the riser equation (3.1).

In the experiments we will assume that the sensors are able to measure the displacement directly. In the real world one would however install accelerometers and rather integrate twice in time to calculate the displacement. This simplification should not affect the reliability in the results when it comes to comparing the different machine learning models and their ability to make predictions on the riser.

Labels

The term *labels* in the machine learning context corresponds to the value of the output data. In the following experiments the labels will be set to the bending moments of the riser by the wellhead. As was mentioned in the introduction, being able to predict the remaining lifetime of a riser more accurately would be beneficial to the oil industry. To be able to calculate the remaining lifetime implies one has to know the accumulated fatigue. The fatigue can be calculated based on the bending moments the riser has experienced during its lifetime.

One does however only need to know the bending moments at the weakest link, not for the whole riser. The most vulnerable place on the riser is the connection to the wellhead

since this is where it will experience the largest bending moments. This is because the riser is clamped[1] to the wellhead. See figure 4.4 to view the typical bending moments.

Given that today's methods for predicting the risers lifetime are quite pessimistic due to the fact that the consequences of an accident can be huge, being able to more accurately predict the lifetime would be very valuable as it might increase production time significantly.

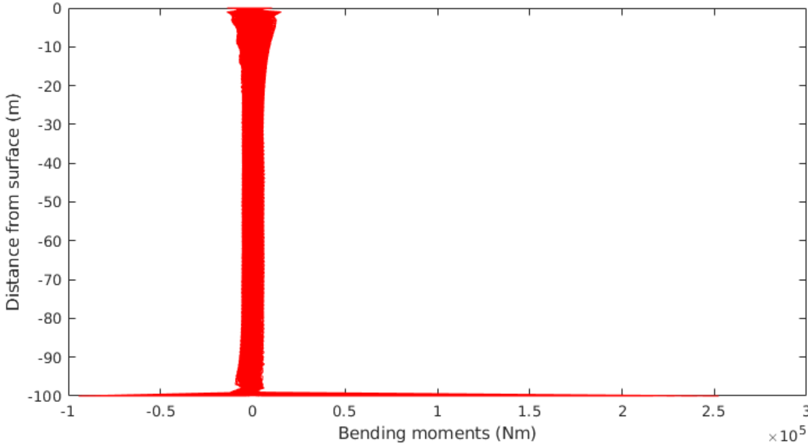

Figure 4.4: Bending moments for a riser of a hundred meters. The largest bending moments can be found where the riser is clamped to the sea bed.

4.2.1 Effects of Δt

The authors of [HR17, CCHC19] argue that since we are looking at a discretization of a dynamical system, the step size Δt should influence the stability of the forward propagation. By considering the forward propagation algorithm for the Standard RNN defined in (3.31) as a discretization of the underlying system of ODEs (3.24), one can derive stability criteria that are dependent on the step size Δt [HR17, Hag00]. In the case of (3.31) we have an Euler discretization of (3.24) which has the stability criterion

$$\max_i |1 + h\lambda_i(J)| \leq 1 \tag{4.5}$$

where $\lambda_i(J)$ is the ith eigenvalue of J and J is the Jacobian of the system of ODEs.

Therefore, to evaluate the effects of Δt, an experiment was designed. The four different RNNs listed earlier were trained on the training set created based on table 4.2. In [BCE$^+$19] they include the step size as a parameter to be trained by the neural network allowing the forward propagation to use adaptive step sizes. In this thesis the step size is however set to be constant.

[1] A clamped beam has constant values in its first and second derivative where it is fastened.

Experiment setup

The experiments were done with 50 fully connected layers each containing 5 nodes. The artificial sensors were placed at $25\,\mathrm{m}$, $50\,\mathrm{m}$ and $75\,\mathrm{m}$ delivering data the same way described earlier. The activation function, $\sigma(x)$, was set to be the hyperbolic tangent function and the hypothesis function, defined as $g(x)$ in chapter 3, was set to the identity function since we are working with a regression model. The objective of the artificial neural networks was to predict the bending moments at the connection between the riser and the wellhead.

The step size of the forward propagation was set to either $\Delta t = 1$ or $\Delta t = \frac{1}{N}$ where N defines the number of layers for all the different networks. The performance of the different networks were tested and evaluated both on the training set with and without noise as well as on the test sets.

The noise was added to the training and test sets after the RNNs were trained. This was done by adding a random normal distributed value to all the data values in the training and test sets with a given standard deviation. The results can be found in the figures and tables that follow.

Results

The theory about reducing the step size of the discretization would increase noise handling and generalization properties seems to hold based on the results from tables 4.3 and 4.4.

	Anti-symmetric	Hamiltonian	Standard	Vanilla
Noise*	-0.226952	-2.820830	-0.587447	-3.675114
Noise**	0.989622	0.979291	0.988476	-3.161441
Training set	0.996650	0.994789	0.997698	0.999491
Test set 1	0.951766	0.929544	0.968854	0.992848
Test set 2	0.994695	0.988832	0.995153	0.998269

Table 4.3: R^2 values for the four different models with step size $\Delta t = \frac{1}{N}$ where N is the number of layers in the RNNs. The Vanilla columns are a reference columns and has no dependency on Δt, but is added to compare the new models to the original one.
Noise*: Standard deviation = 0.1.
Noise**: Standard deviation = 0.01.

	Anti-symmetric	Hamiltonian	Standard	Vanilla
Noise*	-2.708185	-17.327577	-8.659334	-3.675114
Noise**	0.872375	0.806046	0.746500	-3.161441
Training set	0.998741	0.999623	0.999511	0.999491
Test set 1	0.982379	0.994620	0.993748	0.992848
Test set 2	0.997610	0.998912	0.998765	0.998269

Table 4.4: R squared values for the four different models with step size $\Delta t = 1$. The Vanilla columns are a reference columns and has no dependency on Δt, but is added to compare the new models to the original one.
Noise*: Standard deviation = 0.1.
Noise**: Standard deviation = 0.01.

The tables 4.4 and 4.3 contain R^2 values for RNNs made up of N layers with $\Delta t = 1$ and $\Delta t = \frac{1}{N}$, respectively. They give information about how well the different models handle noise and how well they generalizing to new data.

Both settings for Δt have troubles handling absolute noise values with standard deviation being 0.1, but for a noise level with standard deviation 0.01 the value of Δt plays a large role in the accuracy of the predictions.

We do however observe that the models with $\Delta t = 1$ scored higher when predicting on the training set with no noise as well as on the test sets. This comes most likely from the fact that the learning problem is harder for $\Delta t = \frac{1}{N}$ than for $\Delta t = 1$ and therefore the original models were able to train to a higher accuracy. The tables does however not contain information about the models predictions on the test set with noise. To view how the models handles noisy test and training set see figure 4.5 and 4.6

In the tables we also observe that the Standard RNN and the Vanilla RNN seem to be better at approximating the noiseless training and test data for both setting of Δt. This may come from the fact that the two dynamical system based RNNs have different limitations at the price of being able to handle noisy data. In the case of the Anti-Symmetric RNN we fewer degrees of freedom compared to the Standard and Vanilla RNNs because of the anti-symmetric matrices. The Hamiltonian RNN has a disadvantage due to the extra non-linearity because of the matrix product $C(t)^T C(t)$, making it harder to train to convergence than the other RNNs.

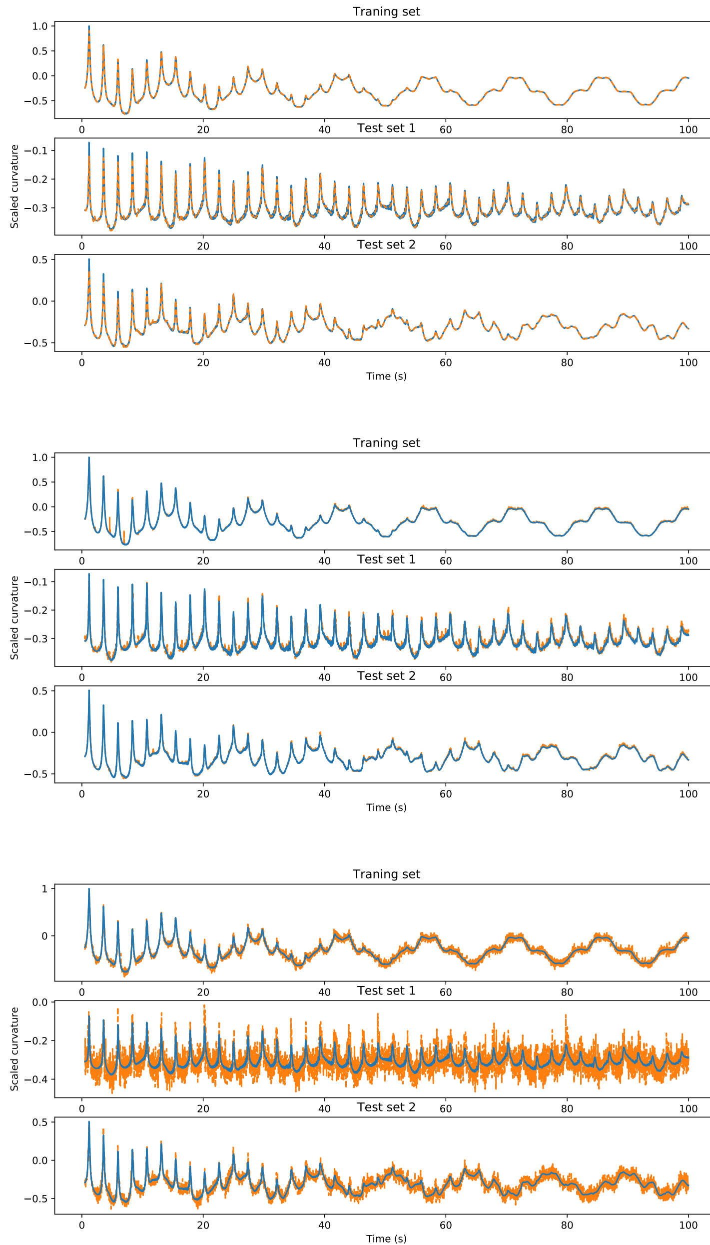

Figure 4.5: Anti-symmetric RNN for predicting riser bending moments with sampling frequency of 100 Hz and using the last 50 states to make predictions. Top figure is the model's prediction on the training and test set with $\Delta t = 1/N$ and no noise. In the middle figure noise has been added and in the bottom figure the same noise has been added to the same model with $\Delta t = 1$

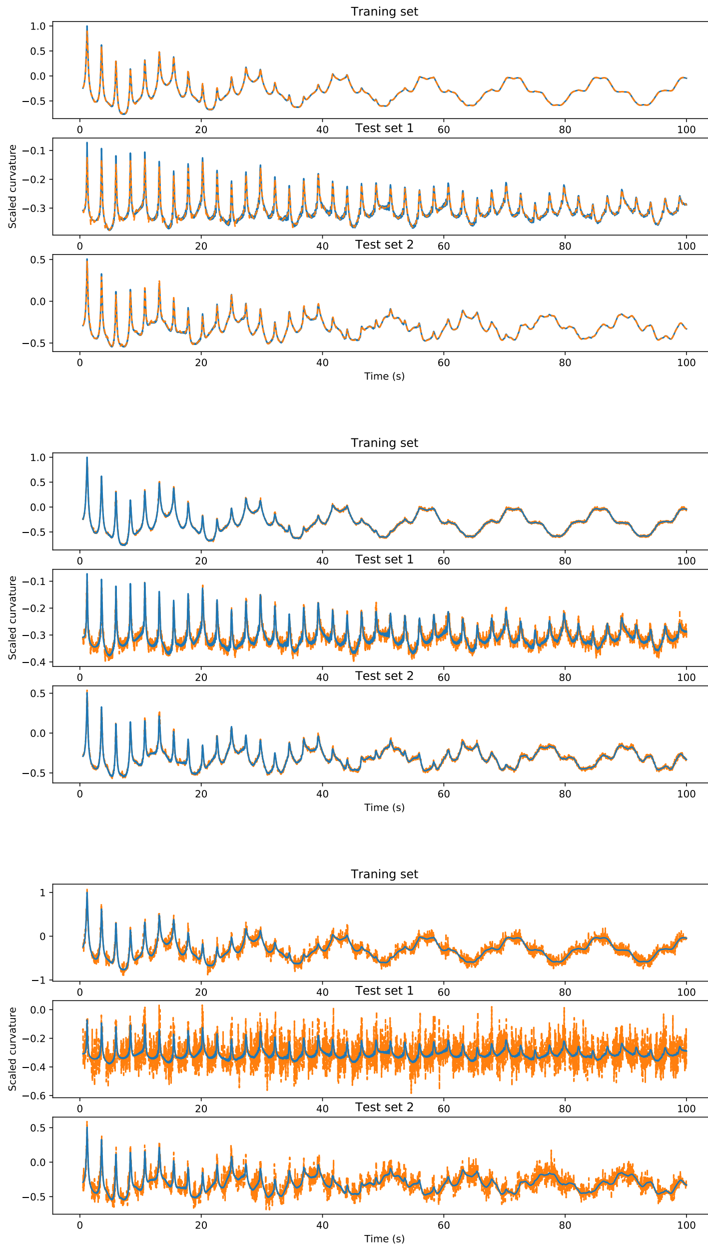

Figure 4.6: Standard RNN for predicting riser bending moments with sampling frequency of 100 Hz and using the last 50 states to make predictions. Top figure is the model's prediction on the training and test set with $\Delta t = 1/N$ and no noise. In the middle figure noise has been added and in the bottom figure the same noise has been added to the same model with $\Delta t = 1$

In figure 4.5 and 4.6 the predictions made on the training and test sets with and without noise for different settings of Δt have been plotted for the Anti-Symmetric RNN and for the Standard RNN. The way the RNNs handle noise can more intuitively be examined in the two figures and confirms what table 4.3 and 4.4 are indicating. The setting for Δt plays a major role in the models ability to handle noise, both on training and test sets. The different RNNs ability to handle noise does however not seem to differ significantly when $\Delta t = \frac{1}{N}$.

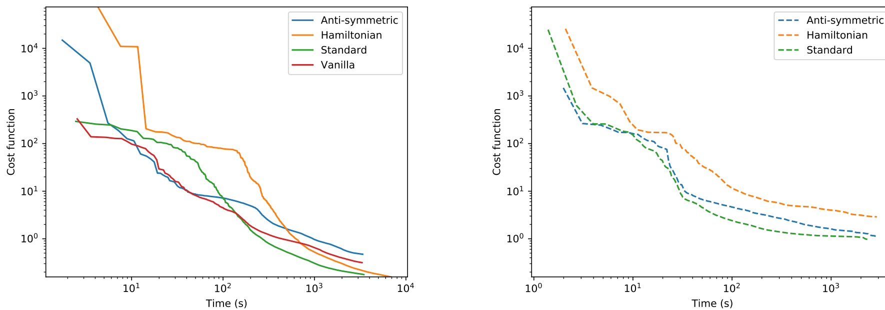

Figure 4.7: The value of the cost functions for the different RNNs as a function of time. Left: $\Delta t = 1$, right $\Delta t = \frac{1}{N}$. We observe that models with $\Delta t = 1$ were able to reach a lower value for the cost function before convergence.

The RNNs with $\Delta t = 1$ were able to reach a smaller minimum with respect to the error function than for RNNs with $\Delta t = \frac{1}{N}$ before terminating. This can be observed in figure 4.7.

Why this happens can be understood by considering the gradients for the Standard, Anti-Symmetric and Hamiltonian architectures given by equation (3.32), (3.33) and (3.34). Since the gradients are multiplied with Δt then the norm of the gradients will be smaller, meaning the model will make smaller steps and reach convergence faster as the convergence criterion is based on the norm of the gradient.

Due to the fact that the Hamiltonian gradient (3.34) is quadratic in Δt while the two other architectures are only linear in Δt we see in 4.7 that the Hamiltonian is not able to reach the same minimum before terminating.

4.2.2 One sensor different positions

The goal of the project is to study how well a machine learning model is able to predict a riser's bending movement based on a few sensors placed over the riser. Until now we have only considered how three equally distributed sensors affect the accuracy of the prediction and evaluated how the model is able to handle noise and new unknown data.

The economically ideal case would be to only have one sensor on the top of the riser and be able to calculate the bending moments by the wellhead. This would be the cheapest option considering installation, maintenance and reliability of the sensor signals. There are however limitations to how well a machine learning model can approximate such complex

dynamics based on a single sensor placed relatively far away from the point of interest. In this section how the position of a single accelerometer affects the accuracy of the model is considered.

Since there was found no superior RNN in the previous section the Anti-Symmetric RNN was chosen. The current goal is to measure the effects of moving the sensors, not comparing the models.

One accelerometer

The experiments were set up with the same training set described in the introduction. Nine different Anti-Symmetric RNNs with 50 hidden layers, each layer containing 5 nodes, were trained where all the parameters were kept the same except for the position of the sensor. The position of the sensor was moved 10 m before training another model. Table 4.5 and figure 4.8 give a description about the accuracy of the models with a displacement sensor at different positions on the riser.

	10 m	20 m	30 m	40 m	50 m	60 m	70 m	80 m	90 m
Training set	0.819	0.845	0.872	0.911	0.953	0.980	0.993	0.997	0.998
Noise*	0.796	0.807	0.835	0.877	0.846	0.972	0.947	0.990	0.983
Test set 1	0.294	0.335	0.392	0.532	0.713	0.825	0.925	0.962	0.982
Test set 2	0.627	0.656	0.726	0.739	0.883	0.954	0.987	0.995	0.995

Table 4.5: R^2 values with one sensor on the riser measuring displacement.
Noise*: Training set with noise. The standard deviation of the noise was set to 0.01.

From table 4.5 we observe that the ability of the model to generalize to new data increases as the sensor is moved closer to the point of interest. The economically ideal position of the sensor was not good for predicting on unseen data, but all positions were able to handle noise well.

Figure 4.8 gives an indication of how the error on the training set evolved as the sensor moved closer to the wellhead. In the same plot the error when noise was added to the training set is plotted. Based on the figure the best spot for placing the sensor with and without noise varies. This fact should be taken into account when attempting to place sensors on a real riser.

Accelerometer and inclination

The inclination of the riser can give valuable information when predicting the bending moments. Additionally, the inclination of the riser at sea level, meaning at the oil rig, can easily be measured using relatively cheap instruments as this sensor does not need to be submerged into water. Using this considerable free information together with some strategically placed sensors along the riser can give better predictions of the bending moments by the wellhead.

A new set of experiments were conducted, similar to the ones described in the previous section. The only difference was that the inclination at sea level was added to the input. The results from the experiments can be found in table 4.6 and figure 4.9

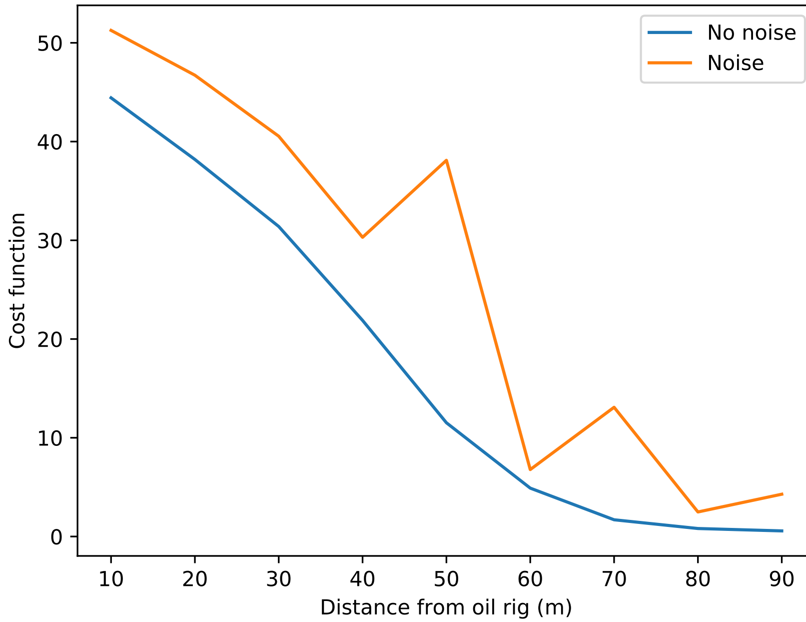

Figure 4.8: The cost function on the training set as a function of the distance from the displacement sensor to the rig. The blue curve is the error on the training set without noise and the orange curve is the error on the training set with noise.

	10 m	20 m	30 m	40 m	50 m	60 m	70 m	80 m	90 m
Training set	0.912	0.950	0.959	0.969	0.980	0.988	0.992	0.997	0.998
Noise*	0.799	0.896	0.942	0.963	0.973	0.975	0.990	0.989	0.983
Test set 1	0.436	0.481	0.563	0.650	0.727	0.876	0.895	0.963	0.983
Test set 2	0.601	0.713	0.800	0.854	0.932	0.974	0.979	0.995	0.995

Table 4.6: R^2 values for one sensor on top measuring inclination and one sensor somewhere else on the riser measuring displacement.
Noise*: Training set with noise. The standard deviation of the noise was set to 0.01.

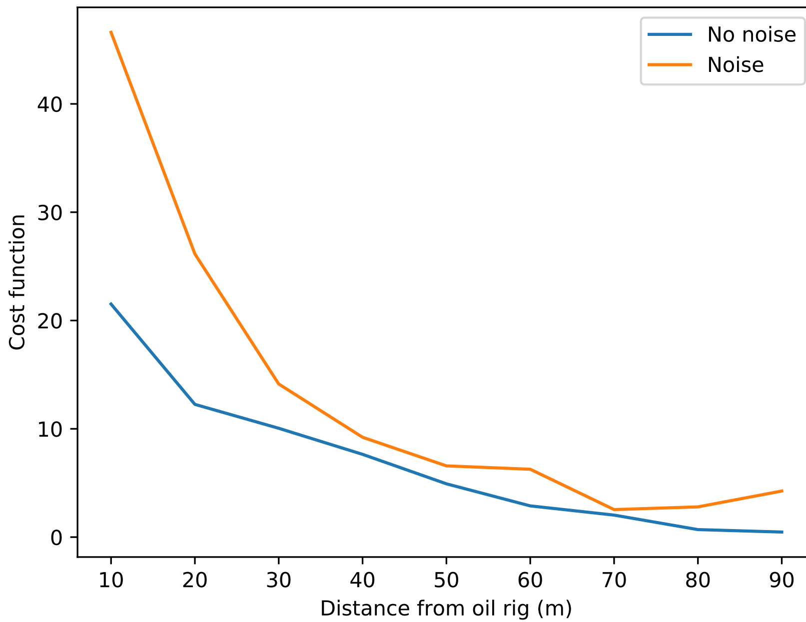

Figure 4.9: The cost function on the training set as a function of the distance from the displacement sensor to the rig. The blue curve is the error on the training set without noise and the orange curve is the error on the training set with noise.

The table 4.6 shows that the effect of adding another sensor to the input gives a higher accuracy model for approximating the data with and without noise. The model was also able to make better predictions on the test sets, but for a sufficiently high accuracy the accelerometer still needed to be quite close to the point of interest.

Figure 4.9 shows how the error function behaves as a function of the sensors position with and without noise. The figure emits a clear trend of the prediction error being reduced as the sensor approaches the point of interest. One should however note that the error function with noise starts to increase from around 70 m. This is possibly because the added noise is not relative to the value of the data. Since the risers movement is more restricted closer to the wellhead the noise that is added to sensors at the endpoints will have a greater impact compared to sensors closer to the center.

4.2.3 Several sensors at different positions

In the previous section the position of a single accelerometer was examined. In this section the economically constraint of only having one sensor will be removed and the machine learning models prediction ability will be studied. In the experiments 3, 5 or 10 artificial sensors will be distributed along the riser either equidistantly or using Legendre or Cheby-

shev nodes. The RNNs will be of the same depth and size as in the previous experiment. Figure 4.10 shows how the nodes were placed along the riser for the different settings.

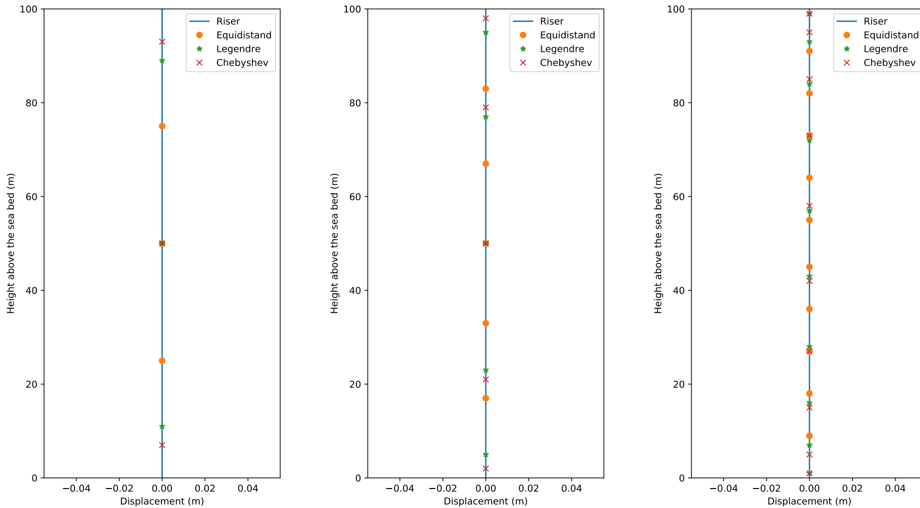

Figure 4.10: Positions for the displacement sensors for the different settings concerning number of sensors and their positions.

The experiments were done in the same fashion as for the experiments with only one sensor. All RNNs except the Vanilla RNN were trained and examined. The following tables contain the results from the different networks accuracy measured using R^2. Noise was also added to the data such that the noise handling ability could be evaluated.

In the tables three different levels of noise was added to the training set, the standard deviation of the noise is indicated by a star (*). One star (*) represent a standard deviation of 0.1, two stars (**) represent a standard deviation of 0.01 and three stars (***) represent standard deviation of 0.001. The caption of the tables describes the number of sensors and how they were positioned. The name of the recurrent neural networks is abbreviated in the following tables to A.S. (Anti-Symmetric RNN), H (Hamiltonian RNN) and S (Standard RNN).

Chebyshev

	A.S.	H	S
Noise*	-2.40	-13.1	-3.6
Noise**	0.946	0.837	0.963
Noise***	0.998	0.987	0.998
Training set	0.999	0.989	0.999
Test set 1	0.991	0.826	0.991
Test set 2	0.998	0.964	0.997

Table 4.7: Chebyshev 3

	A.S.	H	S
Noise*	-6.29	-3968	-8.42
Noise**	-5.98	-0.228	0.382
Noise***	0.991	0.991	0.993
Training set	0.998	0.994	0.999
Test set 1	0.984	0.921	0.987
Test set 2	0.996	0.987	0.993

Table 4.8: Chebyshev 5

	A.S.	H	S
Noise*	-5.14	-10109	-14.9
Noise**	0.901	0.934	-0.170
Noise***	0.998	0.997	0.984
Training set	0.999	0.998	1.000
Test set 1	0.984	0.976	0.997
Test set 2	0.997	0.996	0.999

Table 4.9: Chebyshev 10

Legendre

	A.S.	H	S
Noise*	-1.54	-30.1	-2.37
Noise**	0.981	0.935	0.982
Noise***	0.998	0.992	0.999
Training set	0.998	0.992	0.999
Test set 1	0.984	0.881	0.989
Test set 2	0.997	0.973	0.997

Table 4.10: Legendre 3

	A.S.	H	S
Noise*	-21.8	-37.6	-10.7
Noise**	0.918	0.829	0.919
Noise***	0.998	0.994	0.998
Training set	0.998	0.996	0.999
Test set 1	0.983	0.947	0.985
Test set 2	0.996	0.992	0.996

Table 4.11: Legendre 5

	A.S.	H	S
Noise*	-16.7	-29.2	-33.1
Noise**	0.824	0.882	0.785
Noise***	0.998	0.997	0.997
Training set	0.999	0.998	0.999
Test set 1	0.990	0.976	0.993
Test set 2	0.998	0.996	0.998

Table 4.12: Legendre 10

Equidistant

	A.S.	H	S
Noise*	-0.248	-2.81	-0.652
Noise**	0.990	0.979	0.988
Noise***	0.997	0.995	0.998
Training set	0.997	0.995	0.998
Test set 1	0.952	0.930	0.969
Test set 2	0.995	0.989	0.995

Table 4.13: Equidistant 3

	A.S.	H	S
Noise*	-1.91	-28.9	-0.275
Noise**	0.949	0.953	0.985
Noise***	0.997	0.996	0.998
Training set	0.998	0.997	0.998
Training set	0.975	0.966	0.980
Training set	0.996	0.993	0.996

Table 4.14: Equidistant 5

	A.S.	H	S
Noise*	-2.01	-44539	-18.6
Noise**	0.888	0.777	0.562
Noise***	0.998	0.996	0.995
Training set	0.999	0.998	0.999
Test set 1	0.989	0.989	0.994
Test set 2	0.997	0.997	0.998

Table 4.15: Equidistant 10

In these experiments we observe an upper limit to the number of sensors concerning the ability to handle noise. This comes from the fact that the noise added to the data is absolute, the relative error will therefore be larger as the sensors are moved closer to the endpoints. If the noise is small compared to the data then it has close to no effect on the accuracy on the model.

Again the three different methods did not show any significant differences when it comes to accuracy both on the training set and on the test sets. A possible trend is that the Hamiltonian method was somewhat worse than the two others, which may come from the fact that it is harder to train.

4.2.4 Diffusion constant

In the theory section, when describing the Anti-Symmetric RNN, it is argued that setting the eigenvalues of the Jacobi to be imaginary would result in a system with stable forward and backward propagation.

As an illustration, inspired by [HR17, CCHC19], four phase portraits were made for four different settings for the weight matrices in a Standard RNN. The eigenvalues for the matrices could either be only positive, negative, imaginary or imaginary with diffusion. The matrices could be only one of the following:

$$K_p = \begin{bmatrix} 2 & -2 \\ 0 & 2 \end{bmatrix}, \quad K_n = \begin{bmatrix} -2 & 0 \\ 2 & -2 \end{bmatrix}, \quad K_i = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \quad \text{or} \quad K_{id} = \begin{bmatrix} -\gamma & -1 \\ 1 & -\gamma \end{bmatrix}.$$

The matrices have in order from left to right positive, negative, imaginary and imaginary with diffusion eigenvalues. If one initiates a Standard RNN with N hidden layers, initiates all the weight matrices $W^{(k)}$ to only be one of the listed matrices and recall that the forward propagation algorithm for the Standard RNN is

$$\begin{aligned} a^{(k)} &= a^{(k-1)} + \Delta t\sigma(W^{(k-1)}a^{(k-1)} + V^{(k)}x^{(k)} + b^{(k)}) \\ Y^p &= g(W^{(N)}a^{(N)} + b^{(N+1)}), \end{aligned}$$

then we again observe that the Standard RNN can be viewed as an Euler discretization of

$$\dot{a}(t) = \sigma\left(W(t)a(t) + V(t)x(t) + b(t)\right).$$

The system of ODEs has the following phase portraits for the four different matrices when $x(t)$ and $b(t)$ are zero

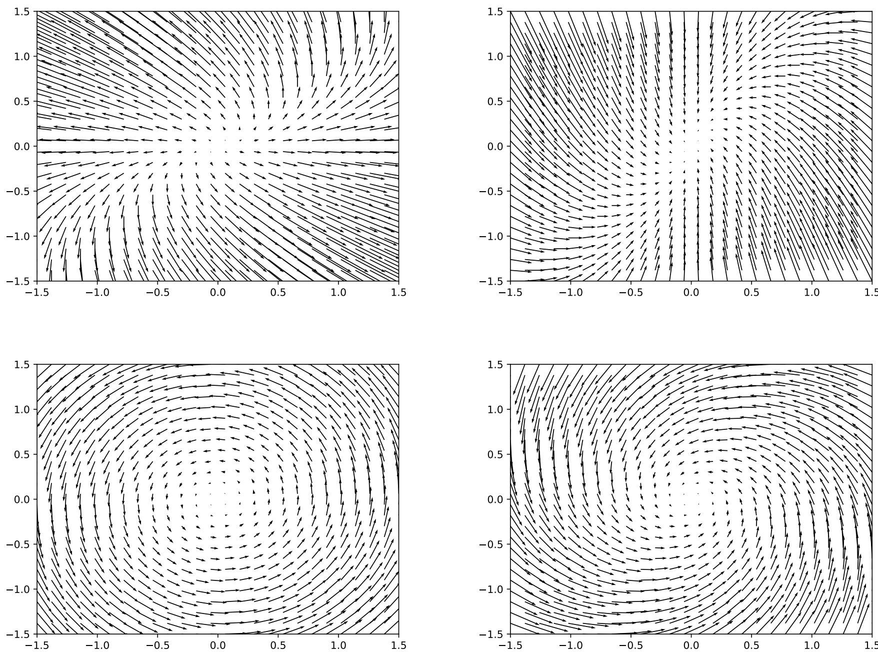

Figure 4.11: Phase portraits for the underlying dynamical system of ODEs belonging to a Standard RNN initiated with transformation matrices with either positive (t.l) negative (t.r.) imaginary (b.l.) or imaginary with diffusion (b.r.) eigenvalues

In the theory section it was argued that positive eigenvalues would mean that small changes in the input data could lead to large deflections in the output. The same effects can be found in the top left phase portrait in figure 4.11. For networks with negative definite matrices we see the opposite effect as input data is pushed towards a stable equilibrium point. This will make the predictions stable, but the learning problem hard since

large changes in the input data will be significantly smaller after being passed through the network.

The two bottom phase portraits are made using matrices with eigenvalues being purely imaginary and imaginary with diffusion. In these phase portraits any change in value will stay close to a stable solution, but the inverse problem is easier to solve due to the fact that the input data does not converge to the same point.

Experiment with the diffusion constant γ

In section 3.2.3 the hyper parameter γ was introduced to add some diffusion to the forward propagation of the Anti-Symmetric RNN. In what follows are the results of an experiment conducted for evaluating the effects the diffusion constant γ has on the Anti-Symmetric RNN's accuracy. The RNNs were set up with $N = 50$ fully connected layers with 5 nodes in each layer. The RNNs were trained to predict the bending moments of the riser by the wellhead based on three equidistant sensors. All the parameters were kept the same except for γ. The results can be found in table 4.16 to 4.18.

	-0.1	-0.15	-0.2	-0.3	-0.5	-1.0	-5.0	-10.0
Noise*	0.220	-0.560	0.318	0.054	-0.456	-0.282	-0.165	-8.25
Noise**	0.992	0.978	0.991	0.990	0.987	0.987	0.986	0.838
Training set	0.997	0.997	0.997	0.997	0.996	0.996	0.997	0.994
Test set 1	0.963	0.963	0.954	0.951	0.948	0.944	0.950	0.935
Test set 2	0.993	0.993	0.995	0.995	0.993	0.994	0.994	0.990

Table 4.16: Output with negative diffusion constants, the two top rows represent the training set with added noise.
Noise*: Standard deviation = 0.1.
Noise**: Standard deviation = 0.01.

	0
Noise*	0.140
Noise**	0.992
Training set	0.997
Test set 1	0.951
Test set 2	0.994

Table 4.17: Output with no diffusion constant, the two top rows represent the training set with added noise.
Noise*: Standard deviation = 0.1.
Noise**: Standard deviation = 0.01.

	0.1	0.15	0.2	0.3	0.5	1.0	5.0	10.0
Noise*	-0.267	-0.077	0.493	0.360	0.325	0.458	0.280	0.235
Noise**	0.987	0.992	0.993	0.986	0.989	0.993	0.991	0.990
Training set	0.996	0.997	0.997	0.996	0.996	0.997	0.996	0.995
Test set 1	0.949	0.954	0.954	0.950	0.955	0.974	0.956	0.937
Test set 2	0.993	0.995	0.994	0.993	0.992	0.994	0.993	0.989

Table 4.18: Output with positive diffusion constants, the two top rows represent the training set with added noise.
Noise*: Standard deviation = 0.1.
Noise**: Standard deviation = 0.01.

The most obvious effect of γ can be found in the first and second row of the two preceding tables. If γ is too negative then the models ability to handle noise is drastically reduced. This is due to the fact that the matrix in the Anti-Symmetric RNN is defined as

$$C(t) - C(t)^T - \gamma I.$$

If the third term is too positive then small changes in the data such as noise will be amplified in the output.

Empirically the best choice of the diffusion constant γ lies within the range $[0.2, 1]$ for this specific setup of the Anti-Symmetric RNN. Good values for γ is highly dependent on the depth and size of the system along with other parameters such as Δt. Finding rules or heuristics for choosing γ could be the ground for further studies. The RNN could also be implemented such that γ is one of the parameters found by the training algorithm.

The training time for the different RNNs also varied with respect to γ. A common trend is that when the value of γ got large in absolute value the network became harder to train. This seems to come from the fact that we are experiencing exploding or vanishing gradients. A logarithmic plot of the training times can be found in figure 4.12.

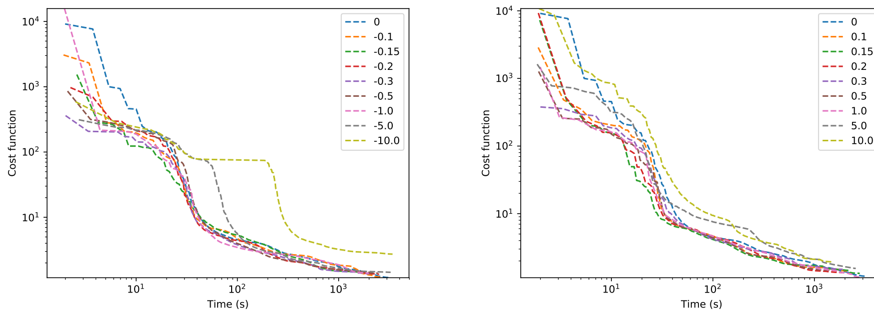

Figure 4.12: Training time for the different diffusion settings, the legend gives information about the value of γ.

4.2.5 Abilities of regularizers

Regularizers are used as a tools for avoiding overfitting [SHK+14]. In this sub-section the effect the regularizers introduced in chapter 3, the weight decay (3.36) and the custom dynamical system based regularizer (3.37), will be evaluated. A control will also be evaluated with no regularizer added to the error function.

All the experiments were done with RNNs consisting of 50 hidden layers where the size of each layer was set to 5. The RNNs were trained to predict bending moments of the riser by the wellhead based on 3 equally distributed sensors. The networks were randomly initialized and trained until convergence or to a predefined maximum iterations.

The results from the different RNNs with varying settings for regularizers can be found in tables 4.19 to 4.21. Additionally, noise was added to the training set the same way as before and the RNNs noise handling ability can also be viewed in the tables.

	Anti-symmetric	Hamiltonian	Standard	Vanilla
Noise*	0.1626	-0.5743	-0.6387	-7.8510
Noise**	0.9932	0.9767	0.9815	-4.8364
Training set	0.9970	0.9955	0.9978	0.9996
Test set 1	0.9634	0.9390	0.9729	0.9954
Test set 2	0.9938	0.9904	0.9951	0.9974

Table 4.19: Dynamical system based regularizer (3.37), the two top rows represent the training set with added noise.
Noise*: Standard deviation = 0.1.
Noise**: Standard deviation = 0.01.

	Anti-symmetric	Hamiltonian	Standard	Vanilla
Noise*	-0.1752	-0.9604	-0.1994	0.0468
Noise**	0.9583	0.9508	0.9590	0.9615
Training set	0.9700	0.9674	0.9707	0.9710
Test set 1	0.8582	0.8400	0.8604	0.8422
Test set 2	0.9478	0.9452	0.9490	0.9503

Table 4.20: Weight decay (3.36), the two top rows represent the training set with added noise.
Noise*: Standard deviation = 0.1.
Noise**: Standard deviation = 0.01.

	Anti-symmetric	Hamiltonian	Standard	Vanilla
Noise*	-3.0585	-0.6495	-0.4935	-96.6314
Noise**	0.7213	0.9707	0.8529	0.3919
Training set	0.9998	0.9991	0.9998	0.9997
Test set 1	0.9962	0.9782	0.9962	0.9709
Test set 2	0.9939	0.9878	0.9943	0.9894

Table 4.21: No regularizer, the two top rows represent the training set with added noise.
Noise*: Standard deviation = 0.1.
Noise**: Standard deviation = 0.01.

As we would expect the methods are better at predict on the training set when there is no regularizer present. All of them do pretty bad with standard deviation being 0.1, with standard deviation being 0.01 the custom regularizer is better than weight decay which is slightly better than no regularizer. Still a good job is done at adapting to the new unknown data.

From the graphs in figure 4.13 we observe that the networks with no regularizer were able to reach lower error values on the training set as opposed to the networks with weight decay regularizer or the dynamical system regularizer. This is due to the fact that the optimization problem becomes harder when there are regularizers present.

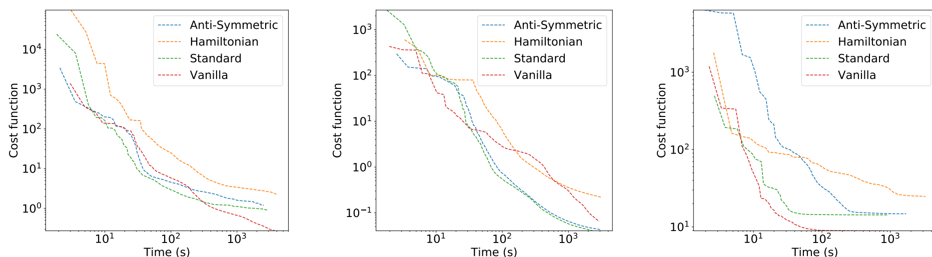

Figure 4.13: The figures show how the cost function is reduced as a function of time during training with the different regularizers. The figure to the left uses the custom regularizer for the dynamical system based neural networks. The middle figure uses no regularizer. The figure to the right uses the standard weight decay regularizer.

4.2.6 Optimization

The different kinds of optimization algorithms defined in the theory section will be evaluated. As mentioned in [HR17] the type of optimization methods greatly affect the networks ability to train as well as how it might generalize later.

In the previous experiments the BFGS method Cholesky decomposition was applied. This is because this was the method which proved to the most efficient and reached the highest accuracy on the RNNs we were training. The Adam method was also considered, but due to the difficulty of adding a regularizer this method was rejected. However, given

the success this method has had with standard weight decay, adding a custom regularizer to the Adam method could be the ground for further studies.

Additionally the stochastic Gauss-Newton method was evaluated, but was also rejected due to the superiority of the BFGS with Cholesky decomposition for smaller neural networks. If, however, the stochastic Gauss-Newton could be properly parallelized then it would be the method of choice due to the relatively fast convergence. It is also able to handle larger artificial neural networks than the BFGS methods which will eventually run into memory problems due to storing the Hessian approximation.

In the first experiments we consider only the convergence rate of the different optimization methods with respect to time. Four Anti-Symmetric RNNs of different sizes were trained with the different optimization algorithms. The prediction error as a function of the time spent training the model can be seen in figure 4.14.

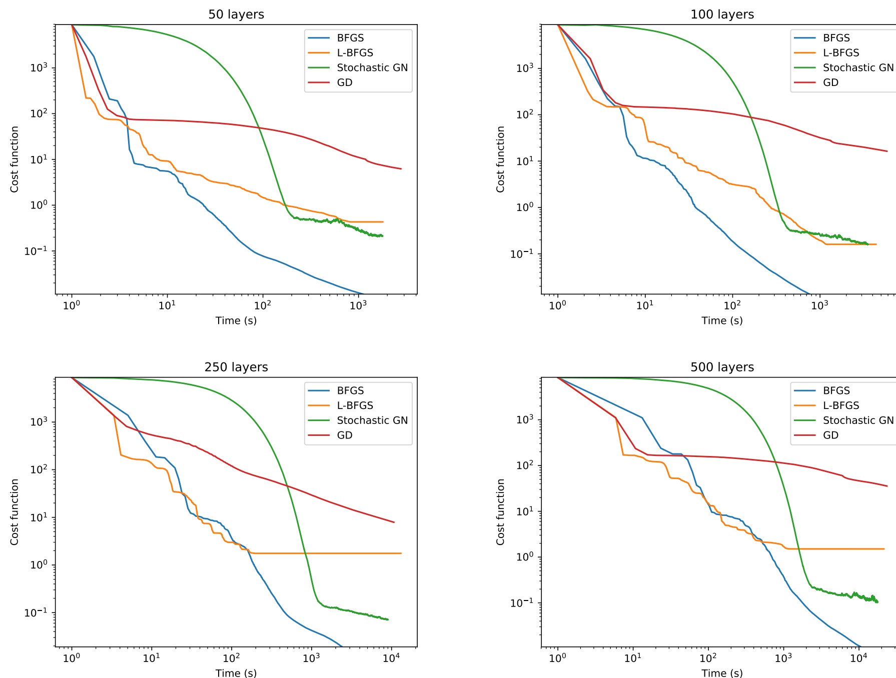

Figure 4.14: The figures shows the value of the cost function as a function of training time. Four of the optimization algorithms explained in section 3.3 were used to train the Anti-Symmetric RNN with four different depths. The number of layers are given by the figure titles.

We observe from figure 4.14 that the BFGS method with Cholesky decomposition performs better than the other optimization algorithms. The stochastic Gauss-Newton could however be better parallelized, indicating that it could be the method of choice with a better implementation.

Lastly, in the theory section a method for updating the Cholesky decomposition of the BFGS matrix was introduced. There it was claimed that the method could solve the linear

system faster and therefore needed less time of finding a minimizer.

Again the Anti-Symmetric RNN was trained on the riser data using the two different optimization algorithms. The following figures show the convergence rate of the two methods as a function of time and as a function of iterations. Observe that they essentially have the same convergence as function of iterations, but not in time. This confirms that the BFGS with Cholesky decomposition has the same updates as the standard BFGS, but is able to calculate them faster as was proposed in the theory section.

The two curves in the different plots in the bottom row of figure 4.15 are not identical. This is because we are working in finite precision and so we will have numerical errors in the iterative solution of the standard BFGS matrix as well as in the solution of the triangular system. This opens for the possibility of the two methods taking slightly different steps.

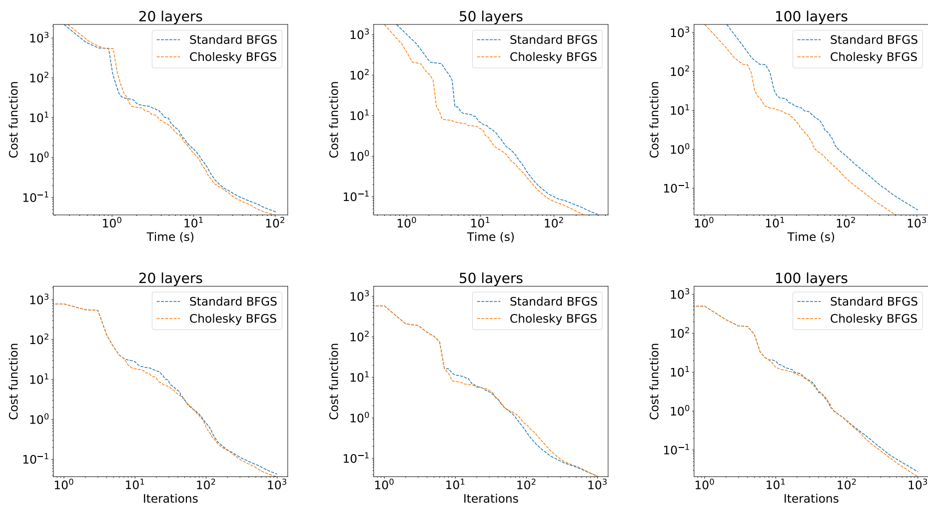

Figure 4.15: The figures show how the value of the error function decreases as a function of time (top row) and iterations (bottom row) for RNNs with 20, 50 and 100 layers. Observe the curves in the figures are different in time, but essentially the same in iterations. Observe that the curves in the figures in the lower row are not identical, this is due to rounding errors of the numerical solver of the standard BFGS matrix.

A logarithmic plot showing the time to complete 1000 iterations as a function of the number of layers in the RNN is given in figure 4.16. Here we observe of the time needed to complete the iterations increases faster for the standard BFGS method compared to the BFGS method with Cholesky decomposition.

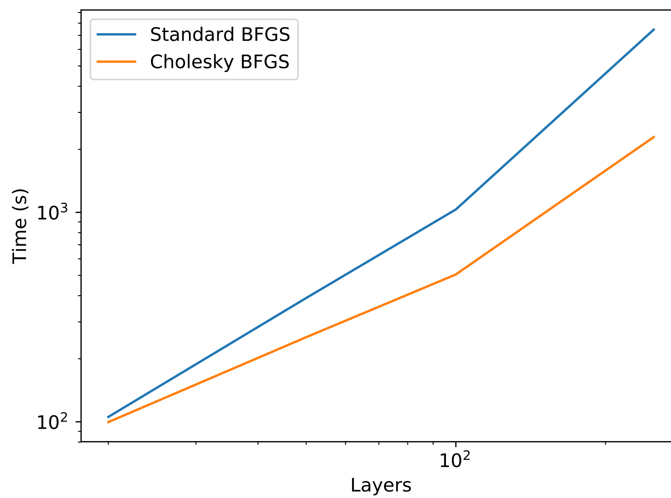

Figure 4.16: The figure shows the time spent for completing 1000 iterations using the standard BFGS method and the BFGS method with Cholesky decomposition for training different Anti-Symmetric RNN. Observe how the blue curve increases faster with the number of layers than the orange curve.

Chapter 5

Discussion

In chapter 4 we first introduced a numerical solver for the riser equations (3.1). This numerical solver is based on solving a one-dimensional PDE in time and space based on a set of parameters given in table 4.1. Even though this PDE contains a small non-linearity this is still a simplified mathematical model.

The numerical solver was created to compute data to be used as features and labels in the machine learning algorithms. The features were based on placing sensors measuring displacement at different positions on the riser. The labels were set to be the bending moments by the wellhead.

We observed that the machine learning models were able to predict the bending moments by the wellhead with a high accuracy if given sufficient amount of sensors or if the sensors were placed close enough to the point of interest. One can therefore draw the conclusion that it will be a trade-off when considering where to place the sensors as well as how many one wishes to use. Having sensors closer to the surface would mean cheaper maintenance, but more sensors would be needed for a sufficient accuracy. Having the sensor(s) placed closer to the wellhead would give better accuracy, but at the price of higher maintenance cost and possibly less reliable sensor data.

There are however other methods that could be used to add more sophistication to the model. One could for instance pass information to the neural network about which sea state it is currently experiencing. Adding information about the sea state could help the RNN make better predictions. As we observed in the chapter 4 it was challenging for the RNN to make accurate predictions when the riser was under the force of small waves. By including the current sea state in the feature set then better predictions should be possible.

The RNNs were able to make quite good predictions with only three sensors. This is however only on a simplified one-dimensional riser. To better understand how ANNs can be used in real life, more complicated mathematical models of the riser should be pursued. This involves increasing the complexity of the governing equations, increasing the number of dimensions and also adding the vortex induced vibrations to the system. If the RNNs are still able to make good predictions and extrapolate to other unknown sea states then such a method would be of high value for the off shore oil industry. Possibly

more sophisticated RNNs should be applied to be able to properly handle the non-linearity of the more complex mathematical models.

In the what that follows are some points of improvement to the machine learning models and the implementation used in this thesis discussed.

5.1 The recurrent neural networks

In the results from the experiments we observed that the Anti-Symmetric and Standard RNN generally performed better than the Hamiltonian and Vanilla RNN. The Vanilla RNN was not able to make accurate prediction on noisy data and the Hamiltonian RNN had trouble generalizing to new data compared to the Anti-Symmetric and Standard RNN.

All the models based on the dynamical system approach were superior to the Vanilla RNN when predicting on noisy data. The step size Δt did however influence their accuracy. This indicates that the dynamical system interpretation proposed by [HR17, CCHC19] could be a valuable point of departure for further theoretical studies.

The reason for the Hamiltonian RNN being inferior to the other dynamical system approaches is believed to come from the fact that the Hamiltonian RNN includes an extra non-linearity, making it harder to train. Additionally the gradient of this RNN is quadratic in Δt implying that when Δt is small the optimization algorithm terminates faster than if it was linear in Δt.

5.1.1 Structure

The fact that the Anti-Symmetric RNN included a weight matrix restricted to be anti-symmetric results in it having fewer degrees of freedom compared to the Standard RNN. However, the Anti-Symmetric and Standard RNNs essentially had the same accuracy for the same training and test data.

A more sophisticated implementation of the Anti-Symmetric RNN needing roughly half the amount of parameters for the same sized network could be performed. This is due to the fact that the each of the anti-symmetric matrices $C^k - (C^k)^T$ can be stored using a triangular matrix. This implies that we need $\frac{n(n-1)}{2}$ parameter to represent an anti-symmetric weight matrix of size $n \times n$. To this end one could increase the size Anti-Symmetric RNN such it has the same number of parameters as the Standard RNN and compare their performance. The results in chapter 4 indicate that this improved Anti-Symmetric RNN could dominate the Standard RNN.

5.1.2 Hyper parameters

The parameter γ

The hyper parameter γ could also be better tuned for the Anti-Symmetric RNN to improve the prediction and noise handling abilities. Choosing the optimal γ for the problem is challenging, however, we observed in section 4.2.4 that γ greatly impacted the RNNs ability to make predictions and handle noisy data. Having a formula or heuristic for choosing γ

would be valuable when creating an Anti-Symmetric RNN. Currently γ is found by trial and error.

The parameter Δt

The hyper parameter Δt proved to be of importance for the stability properties of the forward propagation. This comes from the fact that choosing Δt to be small yields a more stable numerical method for solving the dynamical system and consequently a more stable forward propagation.

A numerical investigation was undertaken in section 4.2.1 to better understand the properties of the parameter Δt, but further improvements to this parameter could be done. In [BCE+19] the step size Δt is chosen by the learning algorithm, which could also be done for the RNNs in this thesis.

5.2 Avoiding overfitting

Overfitting is as previously mentioned a large challenge in machine learning. In what follows comes a discussion about the regularization method used for reducing overfitting as well as other separate overfitting tools.

5.2.1 Regularizer

In section 4.2.5 the RNNs were trained with three different settings for regularizers, either weight decay, the introduced smoothing regularizer or no regularizer at all.

The RNNs based on dynamical system theory showed better results when using the smoothing regularizer introduced in section 3.2.3. This corresponds to the theory about the forward propagation being more stable if the weight matrix changes slowly.

5.2.2 Early stopping

The method known as *early stopping* consists of using the prediction error on the test set as an indication on when to stop the training procedure. For the current implementation the training procedure stops after a certain amount of iterations or if the gradient reaches a predefined tolerance.

At the beginning of the training procedure the prediction error on the test set will decrease similarly to the prediction error on the training set. However, after a certain amount of iterations the prediction error on the test set will reach a lower limit and any further iterations will reduce the networks ability to generalize.

The concept of early stopping is therefore to stop the training as soon as the error of the test set start to increase. One should be aware that the prediction error on the test set will not be monotonically decreasing like the prediction error on the training set. Therefore some sophistication must be added to the implementation of early stopping such that it does not terminate while the total trend of the prediction error on the test set is still decreasing.

5.2.3 Pruning

Dropout

The concept of *dropout*, see [SHK$^+$14] for more details, is to temporarily remove a random set of nodes from the network for each iteration in the optimization algorithm. In this way the chances of certain nodes co-adapting is reduced. This method has showed good results [SHK$^+$14, KSH12] and could be tailored to work for the previously defined RNNs.

Winning lottery ticket

Another pruning algorithm is based on *The Winning Lottery Ticket* theorem [FC18]. The idea behind the theorem is that after a time training the ANN some of the weights in the network will usually be dominating the others, these are the weights who the authors consider picked the winning lottery ticket during the random initialization.

The authors of [FC18] found a training procedure which is briefly summarized in what follows: First randomly initialize the network and train it for a sufficient amount of time. When the first round of training is complete prune the edges with small weights and reinitialize the remaining edges to the previous random initialization. Repeat until the accuracy of the network starts to be reduced.

Following this procedure they found that they could remove up to 90% of the edges without a significant reduction of the accuracy of the network.

Implementing pruning algorithms

If one is to implement these methods then using sums to represent the forward propagation instead of matrices would be advantageous. The dropout method would still be possible to implement using matrices since one would only have to remove the corresponding row and column to the dropped node. For the winning lottery it would however be easier simply removing the indices of the pruned edges from the corresponding sums.

Both these methods would essentially make the training time shorter, but the winning lottery ticket procedure should give the best improvement to the speed if up to 90% of the edges can be removed. The winning lottery ticket will however be hard to implement for the Hamiltonian RNNs due to pruning edges from the matrix product $C^T C$ is challenging.

Even though *dropout* and *the winning lottery ticket* method work in a similar fashion they can not be applied together as one attempts to remove relatively weak edges while the other attempts to keep the edges similar in strength but remove none.

5.3 Training

5.3.1 Parallel implementation

For the size of the RNNs used in this thesis it was found that the BFGS method with Cholesky decomposition was the superior method when considering speed and accuracy. The stochastic Gauss-Newton and L-BFGS methods were both able to reach sufficiently

low values for the prediction error, but they never beat the BFGS method with Cholesky decomposition in the experiments in this thesis.

The stochastic Gauss-Newton algorithm's inferiority to the BFGS method with Cholesky decomposition could be changed by parallelizing the procedure of finding the Jacobi matrix. In the experiments each of the rows in the Jacobi was found in series. Since the rows in the Jacobi matrix are the gradients of the residuals in the randomly drawn batch of data, and since each of these gradients can be found independent of each other, this process can be parallelized. This would give a significant speedup if enough processors are available.

The process of finding the gradient used in the deterministic methods was implemented using the Numpy package in python where the matrix operations already are parallelized.

Running a profiler on the code also revealed that the time spent evaluating the activation function and its derivative uses a significant[1] amount of the training time. Being able to parallelize this process and running it on a graphic card would make the time spent evaluating the activation function and its derivative negligible compared to the time spent solving a linear system or evaluating the gradients.

5.3.2 Coding language

The ANNs were implemented in the Python programming language with a C++ plugin for the implementation of the Givens rotations. Python is easy to read and implement, but has significant amount of overhead and therefore writing part of the code to a lower level language could increase the speed of the training procedure.

[1]When training an Anti-Symmetric RNN with 50 layers of size 5 using the BFGS method with Cholesky decomposition the algorithms spent nearly 30% of the time evaluating the activation function and its derivative.

Chapter 6

Conclusion

In chapter 3 the underlying riser equations were presented along with the theory of their discretization in space using Hermite finite elements of order 5 and using the exponential Euler method in time. In the same chapter the underlying theory of artificial neural networks was introduced and the theory about more recent types of recurrent neural networks based on the theory of dynamical systems was explained. Lastly, an overview of the possible gradient based optimization algorithms were presented.

The theory about the new artificial neural networks in chapter 3 was applied to predict the bending moments of a riser based on the displacement measurements from artificial sensors. The features and labels used when training and testing the networks were extracted from the simulated riser data. Training and accuracy results for the different ANNs were presented in chapter 4. They revealed how the new ANN architectures were able to better handle noise while still being able to generalize similarly to the more traditional architectures. The results revealed that the hyper parameters for the new architectures played an important role and choosing these parameters poorly could lead to worse performance than the already existing architectures.

The ANNs ability to predict on the simplified mathematical model of the riser was evaluated and found to be accurate if the number of sensors and their position were strategically chosen. Further studies about the sensor placement and number could be conducted to increase knowledge about the abilities to the different networks. The experiments could also be performed using more complex riser simulations or even real life data. In addition to the effects the number and positioning of the sensors had on the networks prediction accuracy, we also found that the training and accuracy properties for the recently proposed neural networks were better or similar to the traditional ANNs when it comes to predicting, generalizing and handling noise.

In chapter 5 the ANNs performance were discussed and new experiments were proposed. Later, other methods for choosing the hyper parameters were discussed. Addition-

ally, suggestions for improving the efficiency of the current implementation along with other methods for avoiding overfitting were made.

Bibliography

[Asc08] U. Ascher. *Numerical Methods for Evolutionary Differential Equations*. Society for Industrial and Applied Mathematics, 2008.

[AWBD15] Iwona Adamiec-Wójcik, Lucyna Brzozowska, and Lukasz Drag. An analysis of dynamics of risers during vessel motion by means of the rigid finite element method. *Engineering Structures*, 106(4):102–114, 2015.

[BCE$^+$19] Marting Benning, Elena Celledoni, Matthias J. Ehrhardt, Brynjulf Owren, and Carola-Bibiane Schönlieb. Deep learning as optimal control problems: Models and numerical methods. unpublished, 2019.

[Bot10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg, 2010. Physica-Verlag HD.

[BS08] Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Springer, Berlin, 3 edition, 2008.

[BSF94] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994.

[CCHC19] Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. Antisymmetric rnn: A dynamical system view on recurrent neural networks. *ICLR*, 2019.

[CDHS11] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

[CS88] Jack D. Cowan and David H. Sharp. Neural nets and artificial intelligence. *Daedalus, Artificial Intelligence*, 117(1):85–121, 1988.

[Dav91] W. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, 1991.

[Doz16] Timothy Dozat. Incorporating nesterov momentum into adam. *ICLR*, 2016.

[E17] Weinan E. A proposal on machine learning via dynamical systems. *Springer Verlag*, 2017.

[FC18] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018.

[FP63] R. Fletcher and M. J. D. Powell. A Rapidly Convergent Descent Method for Minimization. *The Computer Journal*, 6(2):163–168, 08 1963.

[Gil18] Philip E. Gill. Lecture 2: Limited-memory quasi-newton methods, January 2018.

[GM72] Philip. E. Gill and W. Murray. Quasi-Newton Methods for Unconstrained Optimization. *IMA Journal of Applied Mathematics*, 9(1):91–108, 02 1972.

[Gus19a] Halvor Snersrud Gustad. Industrial mathematics, specialization project. unpublished, 2019.

[Gus19b] Halvor Snersrud Gustad. A pde model for estimating the lifetime of a riser. To be published in the proceedings of OMAE2019, 2019.

[GW18] Philip E. Gill and Margaret H. Wright. *Computational Optimization: Nonlinear Programming*. Cambridge University Press, 2002-2018.

[Hag00] William W. Hager. Runge-kutta methods in optimal control and the transformed adjoint system. *Numerische Mathematik*, 87(2):247–282, Dec 2000.

[HGHC14] Wei He, Shuzhi Sam Ge, Bernard Voon Ee How, and Yoo Sang Choo. *Dynamics and Control of Mechanical Systems in Off shore Engineering*. Springer-Verlag, 2014.

[HO10] Marlis Hochbruck and Alexander Ostermann. Exponential integrators. *Acta Numerica*, 19:209–286, 2010.

[Hop82] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[HR17] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse problems*, 34, 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[KB15] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[KH92] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 950–957. Morgan-Kaufmann, 1992.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[LBOM98] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[MOJS50] J. R. Morison, M. P. O'Brien, J. W. Johnson, and S. A. Schaaf. The force exerted by surface waves on piles. *Petroleum Transactions, AIME*, 189(4):149–154, 1950.

[MP43] Warren S McCulloch and Walter H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MW05] Borislav V. Minchev and Will M. Wright. A review of exponential integrators for first ordersemi-linear problems, 2005.

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, 2006.

[OM89] P.J O'Brien and J.F McNamara. Significant characteristics of three-dimensional flexible riser analysis. *Engineering Structures*, 11(4):223–233, 1989.

[OMD87] P.J O'Brien, J.F McNamara, and F.P.E Dunne. Three-dimensional nonlinear motions of risers and offshore loading towers. *International Offshore Mechanics and Arctic Engineering Symposium*, 1(6):171–176, 1987.

[PMB12] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. 2013.

[PS95] M.H. Patel and F.B. Seyed. Review of flexible riser modelling and analysis techniques. *Engineering Structures*, 17(4):293–304, 1995.

[Qua09] Alfio Quarteroni. *Numerical Models for Differential Problems*. Springer-Verlag, 2009.

[QV94] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations*. Springer-Verlag, 1994.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature international journal of science*, 323, 1986.

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[RPK17] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations. *CoRR*, abs/1711.10561, 2017.

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition, 2003.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 2014.

[SM11] Endre Süli and David F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, Cambridge, 7 edition, 2011.

[Spa07] Charles Sparks. *Fundamentals of Marine Riser Mechanics*. PennWell Corporation, 2007.

[WRSN13] S. Wiesler, A. Richard, R. Schlüter, and H. Ney. A critical evaluation of stochastic algorithms for convex optimization. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6955–6959, May 2013.

[ZF13] Matthew D. Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. 2013.