Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Viljar Ness

Simulating Ordinary Differential Equations using the Physics-Guided Machine Learning Framework

Master's thesis in Engineering Cybernetics
Supervisor: Adil Rasheed
July 2021

NTNU
Norwegian University of
Science and Technology

Viljar Ness

Simulating Ordinary Differential Equations using the Physics-Guided Machine Learning Framework

Master's thesis in Engineering Cybernetics
Supervisor: Adil Rasheed
July 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Preface

I want to thank Adil Rasheed for his supervision and help in this thesis, as well as introducing me to the concepts of big data cybernetics and hybrid analyses and modeling. I think these are fields of study that can be used in many applications, also outside of the traditional engineering cybernetics field. Creating explainable and efficient models that can combine known dynamics and collected data opens up several possibilities for creating trustworthy predictions for many fields.

- Viljar Ness

Contents

List of Figures

List of Tables

Abstract

The expanding amount of available data from measurements have made data-driven modeling and machine learning popular approaches to modeling. However, these methods often come with the disadvantages of having low explainability and lack generalizability. In this thesis the physics-guided machine learning framework is used to create neural networks which utilizes information from physics-based modeling. These hybrid-model neural networks, called physics-guided neural network in this thesis, are used to predict ordinary differential equations in a simulation setup. The equations that are used for testing are the Duffing equation and the Rayleigh-Plesset equation. The results from this shows that physics-guided neural networks can give higher accuracy and less uncertainty than deep neural networks even with less layers. This gives accurate networks that are less complex with less trainable parameters, and thus increasing the explainability of the models. It is also shown that the characteristics of the ordinary differential equations has effect on how much the insertion of some known equation knowledge affect the performance.

Sammendrag

Den økende mengden med tilgjengelig måledata har gjort at data-drevet modellering og maskin læring har blitt populære metoder. Disse metodene har noen svakheter. De har mye uforklarlighet, og er ikke veldig generaliserende. I denne avhandlingen er fysikk-drevet maskin læring brukt til å lage neurale nettverk som bruker informasjon fra fysikk-baserte ligninger. De resulterende hybrid-modell neurale nettverka, kalt fysikk drevet neurale nettverk i denne avhandlingen, er brukt til å simulere ordinære differensiallikninger. Duffing ligningen og Rayleigh-Plesset ligningen er brukt for å teste nettverka. Resultatene tilsier at fysikk-drevet neurale nettverk har høyere treffsikkerhet og mindre varians enn dype neurale nettverk, selv med færre lag enn det tilsvarende dype nettverket. Dette gir nettverk som presterer bra, samtidig som det er mindre trenbare parametere. Ved å ha mindre trenbare parametere får man nettverk som er mer forklarbare. Det har også blitt gjennomført utforskning på hvordan egenskapene til differensial ligningene påvirker nettverkets prestasjon.

Chapter 1

Introduction

In this thesis networks based on physics-guided machine learning (PGML) are tested to predict and simulate ordinary differential equations (ODEs). The target of the thesis is to create physics-guided neural networks (PGNN) with as few layers as possible that can make accurate predictions with little uncertainty and discuss how these types of networks can create value. Networks for both the Duffing equation and Rayleigh-Plesset equation were created to test performance of the different network architectures.

In this thesis PGML is used when describing the method framework, while PGNN is used when describing the networks created with this framework.

1.1 Motivation and Background

Artificial intelligence (AI) and machine learning (ML) have become widely popular subjects in multiple fields, such as industry, academics, and governing agencies, including the European Union(17). This trend has emerged with the access to more data, in the form of big data, improved computing power and more easy access programming language ML libraries (29). Many fields have seen success in implementing ML algorithms to solve problems and analyze data, and several big companies, including Google, have made investments towards improvements of the field (16). With the growing use of ML there have also been several research topics that have received more attention, such as explainable AI. One of the research topics that is part of this trend is how featured engineering, and physics-based modeling can be combined with ML algorithms to create more reliable and explainable models.

1.1.1 Value of AI

AI has been used to improve algorithms and find patterns in several fields (29). These results can be used to make processes more efficient and solve challeng-

ing computational problems. With more accessible big data databases more and more companies try to use data to make progressive decisions and increase their efficiency. AI has been a popular field for analyzing huge amounts of data (29), because of its ability to find patterns in large data sets. Yet there are more fields that could potential benefit from improved AI models(30), and it is therefore an interesting research subject.

1.1.2 The need for Explainable AI

One of the major challenges of using ML, especially deep learning (DL) algorithms, to fields such as industry regulation, healthcare, and economics is the lack of explainability (17) for data-driven models. Advanced ML algorithms, such as Neural Networks (NN), are viewed as black boxes(27). Some input is sent into the algorithm, which produce some output. The input-output relationship is built through training, but the resulting relationship is not easily explainable and interpretable. This is a problem that grows as the network expands since the number of trainable parameters multiply rapidly(30). This makes implementation in new scenarios uncertain, as there is not a good understanding of the underlying mechanics of the system. This problem arises as the whole concept of NN is the hidden linear relation of parameters. In a safety critical system such as a nuclear power plant the control-unit must be transparent and explainable. Even though a black box-ML based control-unit could contribute to better results in normal operation, it could not be used if it was unknown how it would react in extreme conditions or scenarios that might be dangerous, such as a core meltdown. This example is purely hypothetical to illustrate the importance of having explainable algorithms.

Because of this, explainable AI has been a focus of study for many institutes the last decade (27). This is also one of the focus points of this thesis. The approach to explainable AI in this thesis will be to insert known dynamic of a system to remove some of the trainable layers, which can be exploited to create networks with a smaller number of layers and trainable parameters. This approach introduces known physical models that are been well understood into the system, thus increasing explainability and making the model more interpretable. The goal of this work is to create models that are more generalizable and efficient than what pure physics-based or data-driven methods are separately.

1.1.3 The need for efficient AI models

In advanced simulation systems, such as Digital Twins, it is important to have models that give an accurate prediction with short run time (32) to be able to represent a real life system in real time. One of the problems using deep

neural networks (DNN) in applications like this is their extensive training time, run time and processing resource requirements(10). The number of trainable parameters and depth of network needs to be reduced for these AI methods to be useful in simulations like digital twins and save processing resources in other applications. One way to do this is by having fewer layers. By using the PGML framework this thesis aims to create networks with as few layers as possible, and thus creating minimal number of trainable weights for the network. This will create more efficient network models.

1.1.4 The need for generalizable algorithms

A prediction algorithm that generalizable will be able to be trained and used for several similar cases. It needs to be able to make good predictions on different types of configurations. This is important as the algorithm can then be used on a new set of configurations, without having to be re-developed. Having good generalizable models will also open more up for online learning of the models. Some generalizability was tested with different initial conditions for the differential equations. When a network is constructed for a differential equation it should be able to be used for any initial values, within the existing bounds of the equation.

1.2 State of the art

DNNs used as universal approximators have had success on large complicated data sets(30).Some example of architectures of DNNs that have had success are Bayesian neural networks, that includes the Bayesian rule to include uncertainty (23), and convolutional neural networks, which has had huge success in image classification problems(7). However, the number of trainable parameters quickly grows as the networks become more complex. This makes the model less explainable, increase the run time, and create longer training times for new cases that sometimes takes days to run(12). Using these types of models in a system will both decrease the reliability, and drastically increase the run time of the whole system. The possibility of using known model dynamics to reduce the number of necessary trainable parameters could improve reliability, explainability, generalizability, and run-time of such DNN models.

There are several methods that works towards making the models more explainable. One such method is to use interpretable surrogate functions for local approximations of the complex models(27). Examples of this method are Local Interpretable Model-Agnostic Explanation and SmoothGrad. Drawbacks with these methods are that they have high computational complexity, and thus increases the run time of algorithms that uses them, and that they try to explain

the models by sets of local surrogate functions, and not the whole model. Another method is Local Perturbations (27). This method uses response to local changes in the network to explain the model. This method shares the downside of being computational complex with the surrogate method. There are also Propagation-Based approaches that uses the internal structures of the models to explain model behavior(27). By propagating the prediction using local redistribution from output to input. There are several methods for this propagation, where the drawbacks and advantages with the methods varies. Such as some struggle with gradient shattering or explanations discontinuities. These methods does not necessarily create "true" explanations, but rather axioms(26). The final method of explaining the behavior of a NN that will be discussed here is Meta-Explanations(27). This method identifies general patterns of classifier behavior, and creates representations of the patterns that are identifiable for humans, such as heat maps. Development of these Meta-Explanation methods that gives results that are easily understandable for humans is still an subject being researched on by multiple institutions (27).

There are methods for predictions that do not use ML. Instead, they are based on the physical known models of a system. These in their full complexity are however also very demanding to run and therefore are represented with reduced order models (ROMs) in many applications(30). These models are created as projections of the data simulated by the full function onto a lower dimensional manifold. The challenge with ROMs is that they require full knowledge of the dynamics of the system and will not work on systems that are not well defined.

There are several approaches of using physics to improve NN architectures. One method called Physics-Informed ML which uses physical laws and equations to give constraints to the networks(31). This method uses custom activation and loss functions based on the physical system and laws governing it. While this method has good results in some applications, this thesis will focus more on a different approach, the PGML framework. The difference between these two approaches is that the PGML injects the know information of the dynamics of the system into the NNs as an featured input(30). This input can be fed in at different locations of the NNs. This type of NN architecture has been used to predict lift coefficient of airfoils with great success(30). This method takes explainable models with known behavior into the NN, which create a NN with behavior that is easier to interpreter. However, there is still untested areas for this type of architecture. This thesis will explore the PGML framework being used to recreate ODEs of second order.

1.3 Research objectives and research questions

1.3.1 Objectives

Primary Objective: The primary objective of this thesis is to explore how PGNN architectures perform at recreating second order ODEs through simulations compared to more conventional DNN architectures.
Secondary Objectives:

- To discover how the placement of $g(x)$ influence the performance of the PGNN.
- To uncover how big impact the PGNN architecture has on the number of layers needed in a NN to get adequate predictions.
- To give an overview of how different types of $g(x)$ and $h(x)$ inputs affect the performance of the networks.
- Discuss possible value creation based on the results.

1.3.2 Research questions

To the best of our knowledge there is currently no published work that explores the characteristics of PGNNs used to predict ODE. To this end, the guiding questions governing the research can be stated as:

- Can a PGNN give better results and less uncertainty than a conventional DNN when simulating ODEs?
- Can a PGNN make accurate predictions with fewer layers than a DNN?
- How does placement of $g(x)$ affect the results of a PGNN?
- Which type of $h(x)$ and $g(x)$ input functions have a noticeable effect on the performance of a PGNN?

1.4 Outline of Report

The thesis comprises of the following chapters and content: Chapter 2 which covers the theories used in this thesis; Chapter 3 dissect the concrete methods and the setups used; Chapter 4 presents the results and discuss them; Chapter 5 present conclusion and future work.

Chapter 2

Theory

This chapter covers the theory used in the work of this thesis, as well as theory used for discussion of the results. The chapter will cover the overview of the topics and give a general knowledge of the concepts that are used. Most of the subjects are covered more deeply in the sources. Several of the figures are heavily based on (34) and (32), as they create good illustrations of the covered theory.

2.1 Physics-Based Modeling

Models and equations based on observable and explainable physics have been the leading approach in engineering (32). In this approach the models are based on mathematical equations that are found through research and experiments that explain a certain physical behavior, such as Newtons Laws or Euler equations in fluid mechanics (24). The equations can be solved analytically or numerically for specific cases, depending on the problem. There are also some differential equations that are non-solvable. This kind of approach of using known dynamics can also be used in field with less definitive laws than physics, such as macroeconomics(6)(8), by using mathematical equations that explains the behavior of the system.

The modeled and solved equations for a system do not usually cover all the physics of a system. This is because to fit the explained physics to the system approximations must be done. In complex systems that has multiple physical governing laws interact, information might get lost from simplifications or unknown influence on the system, as illustrated in figure 2.1.1. In implementation of this approach on a complex system there can be unexplained residuals that does not fit with the governing physics equation. Furthermore problems such as numerical instability, computational complexity and errors from unexplained uncertainty are often encountered (32). However, there are several advantages to physics-based models. Firstly they are interpretable as they are based on ex-

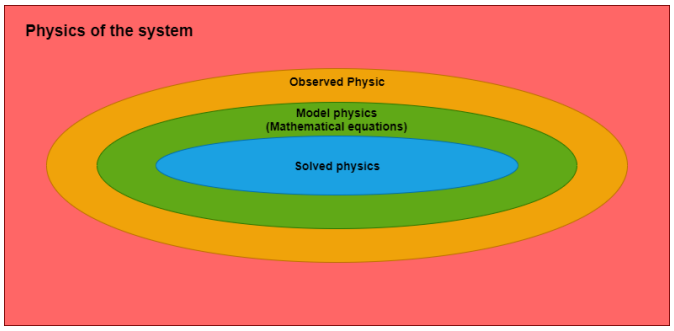


Figure 2.1.1: Physics-Based Modeling: For each layer information is lost due to assumptions and simplifications.

plained physics, and the equations are generalizable to multiple systems with the same physical laws affecting them (14). This makes them trustworthy to use in critical system that requires transparency. Furthermore, physics-based models are in general not affected too much by bias as they are based on physical laws in the form of mathematical equations. Bias can still be introduced by model selection and modeling error, but this bias will be transparent from the documentation of the work. Using the physics-based modeling approach can be challenging with systems that has a lot of factors affecting it, especially when those factors can be uncertain and varying (14). This can lead to models that does not perform well under every circumstance.

2.1.1 ODEs

This thesis will use physics-based ODEs to generate data to test the networks with. ODEs are equations with an independent variable and its derivatives (28). This thesis uses second order ODEs with time derivatives.

2.2 Data-Driven Modeling

Data-Driven Models are models that are based on algorithms finding connections from data and measurements from a system. It has become a popular approach as the amount of available data has drastically grown, improved computer hardware has been developed, and new programming language libraries for creating ML models has emerged (19). This approach creates models based on the data that should represent the whole system, unlike the physics-based approach that only represent the part of the system that is modeled by explainable and testable physical equations. While the physics-based approach goes layer by layer down to the solved equation, shown in figure 2.1.1, the data-driven approach creates a subspace that spawns over all of the system that is

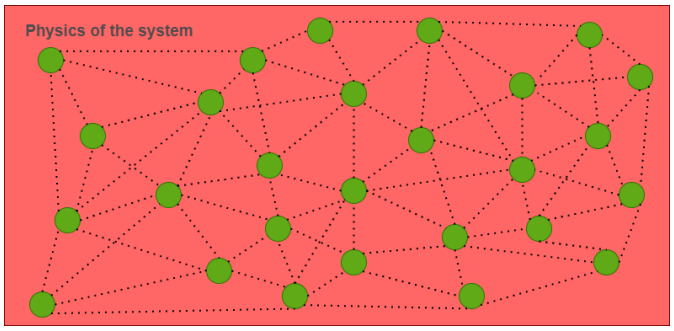


Figure 2.2.1: Data-Driven Modeling: The connection between the data points (The green circles), found through data driven modeling, creates a subspace in physical system.

measured. As shown i figure 2.2.1 the data connection between the data-points, found through data-driven modeling covers a larger spawn of the physical system then the solved physics in figure 2.1.1 does. These models can also improve as more data is given as input to them, as they then get more information about the system. The resulting models are also in general more numerical stable than physical-based models.

Even though the data-driven approach has some advantages, such as not removing information by approximations, it also has some drawbacks compared to the physics-based approach. The training and use of these models are very reliant on a large quantity of high-quality data. Errors in the data can carry into the model making it less reliable. It is important to pre-process the data to reduce run time of the algorithms, to avoid some features being over weighted, and removing biases that might alter the result negatively. Finding correlations between variables, and outlier detection is important parts of the pre-processing (32). In addition, saving all the necessary data in data-warehouses and clouds take up resources and increase energy use, and it is therefore important to analyze which data is important to measure and keep, and not save too much unnecessary data. In this thesis data handling and pre-possessing is not used much as the data is generated from a solver with no noise or other disturbances, and the only parameters are the state value, its derivatives, and time steps. Another challenge is the explainability of data-driven models as discussed as motivation in Chapter 1. This topic will be covered in Section 2.3.

2.2.1 Artificial Intelligence and Machine Learning

The use AI and ML has greatly increased the last decade(19). Both advancement in big data, hardware technology and solid programming libraries have made these fields more accessible and useful. With this growth, and the use of the words as popular buzzwords, the definition of them has become somewhat

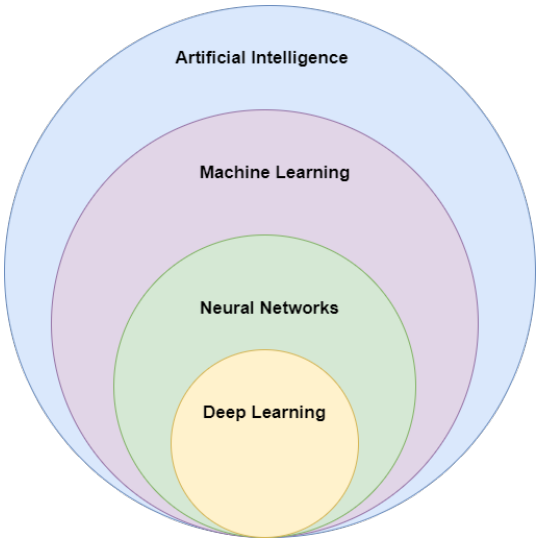


Figure 2.2.2: The hierarchy of AI: This models illustrate the connection between Artificial Intelligence, Machine Learning, Neural Networks and Deep Learning.

blurry in its use. The first thing that will need to be clarified is the difference between AI and ML. AI is a system that can solve complex problems from human understandable input and produce a human readable output. For example speech-to-text technology, where the input is the humans speech, and the output is the text on screen (36).

ML was first coined by Arthur Samuel in 1959 (35).ML algorithms are algorithms that can produce results that are not explicitly programmed by the developer. A ML algorithm has the ability to learn and have behavior outside of the developer's direct control, but still within the bounds of its domain/constraints. In Samuels case it was used for a machine playing checkers. This means that ML algorithms are a subspace of AI algorithms as illustrated in figure 2.2.2. ML is generally categorized into supervised-, unsupervised- and reinforcement-learning(18). Supervised learning (SL) is the method that has been utilized in this work.

SL is learning from a input to a reference output (11). This can be done both in classification and regression problems. The complexity of SL algorithms can go from simple linear regression and logistic regression to complex methods such as artificial neural networks. SL is the ML category that will be utilized in this thesis, with the use of NNs.

2.2.2 Neural Network and Deep Learning

As illustrated in figure 2.2.2 DL is a subspace of NNs which itself is a subspace of ML. DNNs have had a lot of success in recent years, especially in pattern

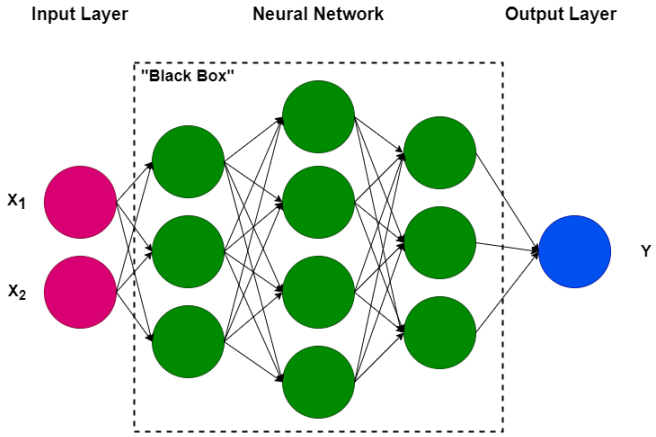


Figure 2.2.3: Neural Network Architecture: A NN with layers consisting of nodes that create the input to output relation. The number of nodes at each layer varies for different model architectures, as well as number of layers.

recognition and classification (10).

A NN consist of layers with nodes, as illustrated in figure 2.2.3, where each layer has a feedforward connection to the next layer. Each node has a weight that is trainable as well as an activation function to get non-linear mapping. The activation function controls when a node is active or not, and how it behaves when active. The process for training the a NNs weights follows (20):

1. All weights are assigned a start value
2. Input is given to the NN
3. The input is transformed to output by passing through the nodes at each layer
4. Output from the network is compared with reference in an error function
5. The error is used to update the weight of each node
6. The error is backpropagated in the network such that each layer minimizes error
7. Repeat step 3-6 until end requirements are met

Backpropagation means that the process propagates starting from the output layer going to the input layer.

There are multiple parameters and algorithm implementation that affects how a NN initiates, how it updates its weights, and when to end training. Stochastic gradient decent optimizer is usually used for the process of tuning the weights. In these optimizers the learning rate decides how much the

weights are changed in response to the error for each iteration. Choosing a learning rate that is too large might result in sub-optimal weights or unstable training, whereas choosing a too small learning rate have the danger of getting stuck in a local minima (20). Furthermore the characteristics of the optimization algorithm used will also affect how well the weights are trained (13).

End requirements can be the number of epochs, where epochs are the number of times the training is done on the training data, and/or it can be early stop for when the algorithm has not improved its loss or accuracy for a given number of epochs.

As for starting weights they are made from a random function around zero, where a seed will decide their start value, and can change the result of the algorithm. One reason for this stuck at a local minima if the starting weights places it close to, or in, that minima (21).

2.3 Explainable AI

The complexity and nested non-linearity of the ML models makes them be considered black boxes, where an input is given, and output is produced, but the relationship of in-to-out is not explainable. This makes it unreliable in safe critical applications, and applications where data privacy requires the algorithms working on the data to be transparent (27). This has led to research into explainable AI, which works on developing methods of explaining, through visualizations or other methods, how the complex algorithms work, such as surrogate functions (27), or methods that tries to replace parts of the models with more explainable modules, such as PGML (30). By creating more explainable AI it is possible to implement AI and ML methods in more situations and create value at several fields(27).

One of the important topics that explainable AI tries to unravel is to identify correlation versus causation in an ML algorithm. The image classification method that won PASCAL VOC was later shown to recognize boats by the presence of water, and horses by the presence of copyright watermarks (22). If a crucial application has correlations like this it can have dire consequences. Algorithms with these kind of correlation problems are also not very generalizable and will not be able to perform well on new applications. For the PASCAL VOC wining algorithm it would not work in a real time application identifying horses from photos, since these would not have the watermarks.

Methods such as Surrogate functions, Local Perturbations, Propagations Based methods, Meta-Explanations (27) and Physics-Informed Neural Networks (31) have made strides through improving explainability of AI in recent years. However there are still challenges to improve explainablity and utilize hybrid models (27) (30).

2.4 Big Data Cybernetics and Hybrid Analysis and Modeling

Big data cybernetics (BDC) was first coined by the Norwegian University of Science and Technology (25). BDC is a framework that combines principles from data-Driven modeling, physical-based modeling, and big data. Instead of representing data with only the black box Data-Driven models from big data, it utilizes physical-based information to make hybrid models. An adaptation from (32) will be used here, and is represented in figure 2.4.1. As shown in this figure the goal of this framework is to try to model as much of the information as possible with the most explainable models. The first step is to fit the physical-based model to the corresponding data. Since this is a model with approximations there will be residuals that are not explain by the physical model that is left over from this model fit. These residuals will then be modeled using explainable data-driven approaches. From this step there can still be residuals that are not interpreted by the explainable data-driven approaches. It is at this stage that black-box algorithms that are not explainable are used to model the final residuals. By doing this order of modeling one use explainable methods on as much of the system as possible, making the whole system more trustworthy and explainable, which is a better alternative than to use black-box methods for all of the data, or only physical-models that does not cover all of the dynamics. This framework works towards removing weaknesses of both the physical-based approaches and the data-driven approaches by using a combination of the two methods together with big data.

A method of combining physics-based and data-driven modeling with big data is called Hybrid Analysis and Modeling (HAM) (34) (32). As figure 2.4.2 HAM combines the different approaches and big data to create a method where the strengths from each of the fields are combined, and thus removes some of the shortfalls of each approach in isolation. There are multiple approaches to implement HAM(34):

- Complete replacement of equations with DNNs
- Modeling the unknown using DNNs and imposing sanity check using equations based on known physics
- Memory embedded reduced order modeling
- Physics / knowledge / regulations informed machine learning
- Dissecting DNNs

This thesis will focus on a similar method to physics/ knowledge/regulations informed machine learning in the form of PGML, which is covered in Section 2.5.

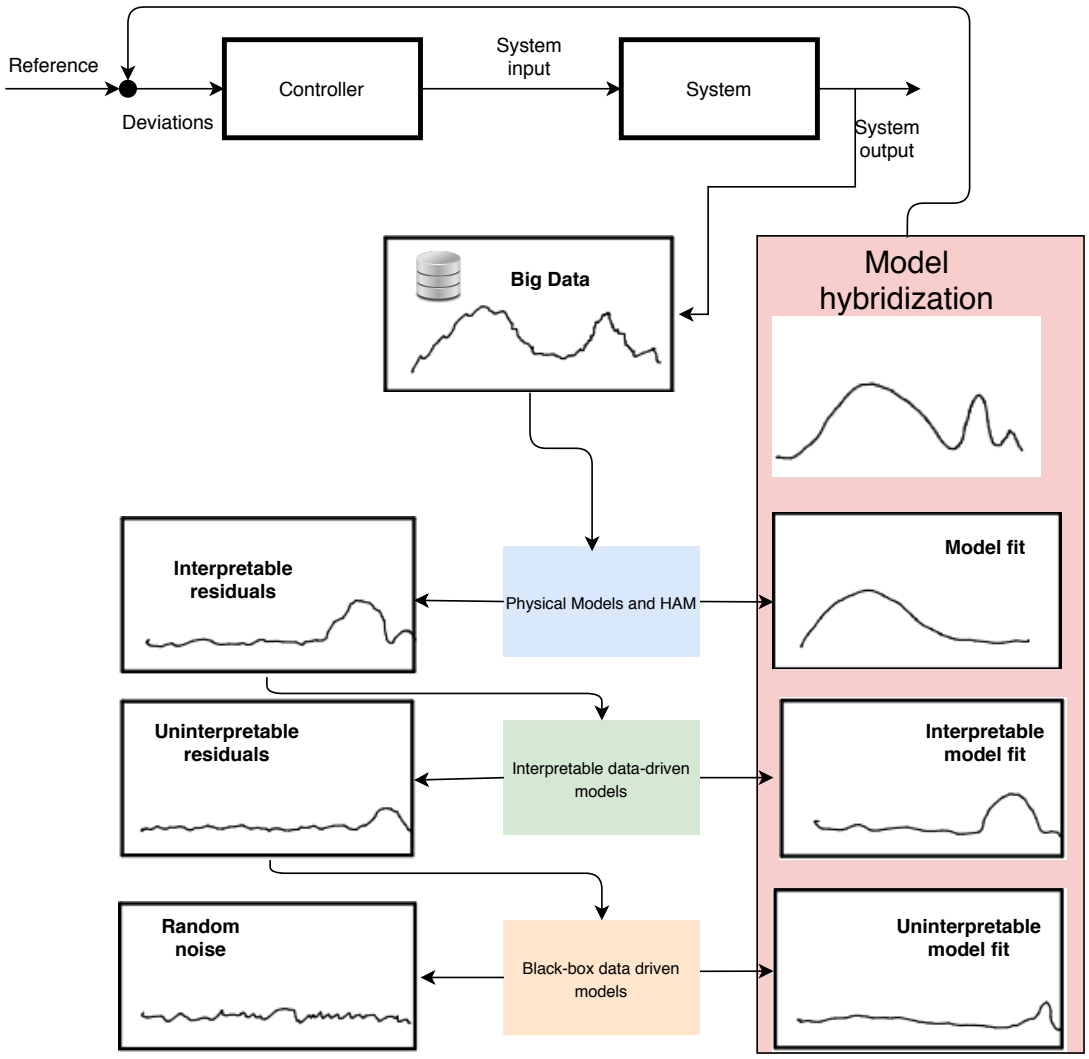


Figure 2.4.1: Big Data Cybernetics: Illustration of the workflow in Big Data Cybernetics in a feedback loop. The measurement data is first fitted to a known physics-based model to represent the main bulk of behavior. In the next step the residuals from the first step are analyzed and fitted using explainable and transparent data-driven modeling. The final step of modeling on the leftover residual are done by black-box methods such as NNs. The combination of physics-based models, explainable data models, and black box data models are then combined to a Big Data Cybernetics Model, which is connected back to the feedback loop.

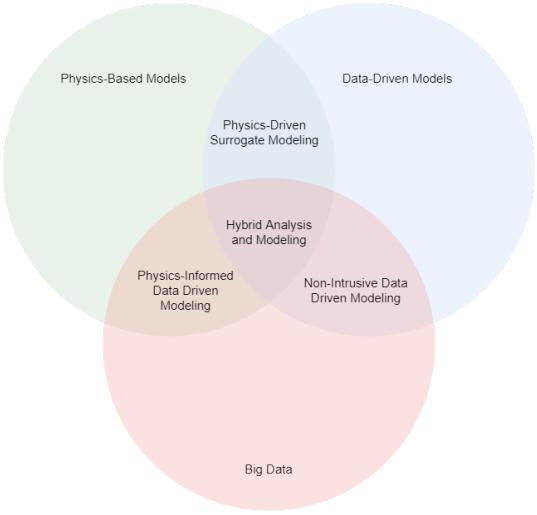


Figure 2.4.2: Hybrid Analysis and Modeling: Hybrid Analysis and Modeling combines the use of Physics-Based Modeling, Data-Driven Modeling and Big Data. This figure also shows other combinations of the fields.

2.5 Physics-guide Machine Learning

Physics/knowledge/regulations informed machine learning works by programming information of the system into a NN (34). One way this can be done is by regularize the cost function using residual of the governing equation. Another method is to insert information from known dynamics of the system into the network. This framework is proposed in (30). This method takes known dynamics of the system and use it to improve the learning process of the NNs. The information can be inserted at the input layer, output layer, or between layers in the NN. From the function 2.5.1 of a system, parts of it such as $h(x)$ and/or $g(x)$, will be added as input somewhere in the network, as shown in figure 2.5.1. The output to input function will then be on the form seen in function 2.5.2, where y is the output, referring to $\dot{x}$, $g(x)$ and $h(x)$ is the same as in function 2.5.1, $\mathcal{F}$ represents the nodes and layers in the NN from where $g(x)$ is inserted, and $\mathcal{N}$ is the NN before $g(x)$ is inserted. In this function there is no unknown information about $h(x)$ which lets it be place at the end of the network, since there is nothing that the network has to learn about it.

$$\dot{x} = f(x)g(x) + h(x) \tag{2.5.1}$$

$$y = \mathcal{F}(g(x) + \mathcal{N}) + h(x) \tag{2.5.2}$$

The PGML framework constraints the NN and encourage it to learn in a manner that is consistent with the known information about the model. This inclusion of the physical-based model in the NN makes the behavior of the

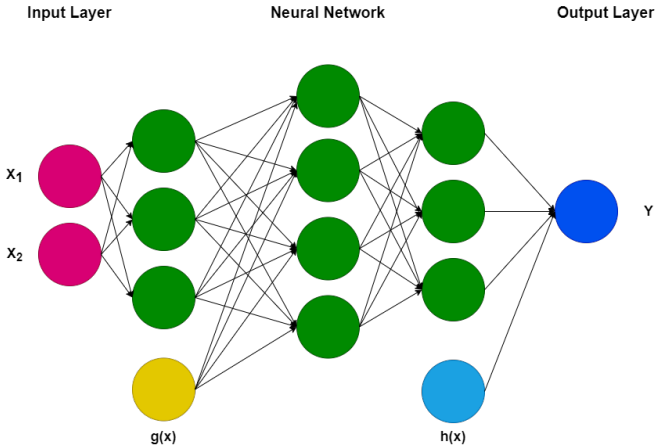


Figure 2.5.1: Physics-Guided Neural Network: Known information of a system is introduced into the network at various stages, creating a network that consist of both trainable weighted nodes in layers and information from the known dynamics.

NN more predictable, and also makes it more explainable as parts of the NN model is based on known, and explainable dynamics, and thus removing some of the trainable parameters which are not easily explainable (30). PGNNs are NN that uses the physics-guided principle. Figure 2.5.2 shows the workflow of the PGML with the contribution from both data-driven method and physics-based methods.

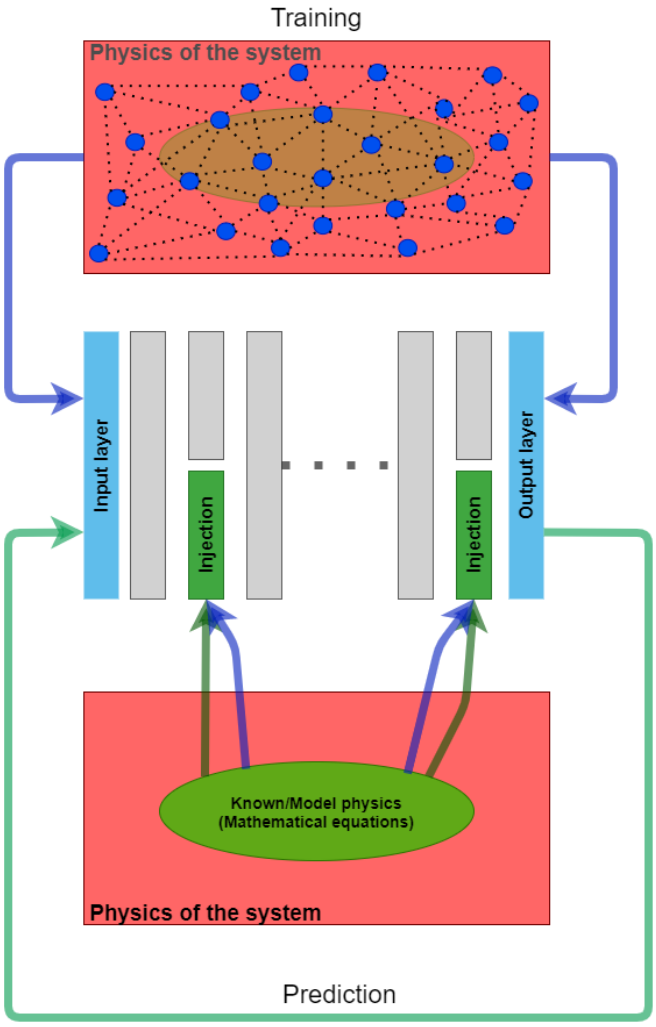


Figure 2.5.2: The workflow of PGML: The measured data is used as input to NNs at training and initial stage. Among the hidden layers there is input from the known physics, both in training and when doing predictions. The output is referenced against measurement. After training the output is used as new input in the prediction stage.

Chapter 3

Method and Set-Up

This chapter will cover the method and set-up that are used to generate the results used in this thesis. First it will cover the equipment and programs used, this will present the computer pacification and programs used. In method and program set-up the method of build and testing the experiments are described. The final part of this chapter covers the build and parameter values for the ODEs and the networks.

3.1 Equipment and Programs

This section covers the equipment and programs used, which in this works will be the computer specifications, and programs, including versions, used. All these can affect the results for the different networks.

3.1.1 Python 3.8

All code used for this thesis is programmed in Python 3.8. Python is an open-source object-oriented high-level programming language with thousands of libraries for different functionalities (1).

The following libraries for Python (aside from the base Python 3.8 libraries) has been used here:

- numpy 1.19.5
- matplotlib 3.3.4
- tensorflow 2.4.1
- scipy 1.6.2

Numpy has been used for handling use for numerical values, arrays, list, tuples, and other types of structures for saving and handling numbers. Matplotlib has been used for all plotting. TensorFlow was used for building, training ,and using the NNs. Scipy was used to solve the differential equations for generating data.

TensorFlow

This thesis uses TensorFlow for Python as its libraries for creating NNs. TensorFlow is an open-source software library developed by the Google Brain Team (5). TensorFlow was chosen as the machine learning library in this thesis based on the implementation of PGML in (30), and its corresponding github repository (2). Other possible choices for machine learning libraries, such as PyTorch, was not explored.

3.1.2 Computer specifications

All of the test were run on the same computer with following specifications:

- CPU - Intel Core i7 8750H 2.2GHz
- GPU - NVIDIA GeForce GTX 1060 8GB
- Motherboard - LENOVO LNVNB161216
- RAM - 2x Samsung M471A1k43CB1 DDR4 8GB
- Windows 10 64 bit

The networks were not run on the graphics card due to difficulties with instalment of TensorFlow for GPU.

3.2 Method and program set-up

For constructing the tests the following steps were followed for each equation:

1. Create a function that generates the differential equation.
2. Use the function to create different lengths of the equation with different initial values (size of the time step, δt, was kept constant).
3. Plot the results of the differential equation function to make sure the results are feasible, and avoid singularities and instabilities.
4. Build, train and test a DNN and a PGNN with a single seeding.

5. Tune learning rate, number of epochs and depth of the networks and repeat step 4 until the test results do not diverge and give a reasonable results, in the value range of the original equation.

6. Build, train and test the networks with 30 different seeds.

7. Repeat step 6 with different configurations of $g(x)$ and $h(x)$ for the PGNN, and different number of layers for both network types.

The reasons for checking the differential equations and avoiding instabilities is that from early testing it was found that sometimes the solver for the differential equation did not give reasonable results, which might be caused by a numerical instability. These errors would then be carried down into the training of the NNs and add potential error in the training process. Furthermore points of instability would create *inf* values, which in turn would create problems when being used as an input to a NN.

3.2.1 Tuning the networks

The training and validation loss graphs were studied when tuning the networks. Epochs would be added to the networks training cycle if they seem the loss values did not converge around some value that was lower than start value. Otherwise, the learning rate or depth of the networks was changed in a combination of test and fail and finding similar cases where NNs had been used and study the approaches used there. Some of the parameters are just roughly tuned and it has not been a focus point to optimize these in this thesis, but rather get them good enough to analyze the difference in performance between DNNs and PGNNs. Parameters such as learning rate, validation split, shuffle, batch size, optimizer and loss function was kept the same for both the DNNs and PGNNs in a run to not give them different initial conditions of performance outside the insertion of known dynamics and number of layers.

As discussed in Chapter 4 the tuning for the Rayleigh-Plesset equation did not manage to get accurate results. The reason for the error was not found, and after a while was not looked further into as the results still showed some interesting differences in performance for the different architectures.

3.2.2 Workflow of code for testing the performance of the NNs

Figure 3.2.1 shows the workflow of the code that from generating the data, to training and testing the networks, and finally returning the interesting results. The training data contains multiple sets of generated equation data with different initial values, and the testing set was created with a unique set of initial

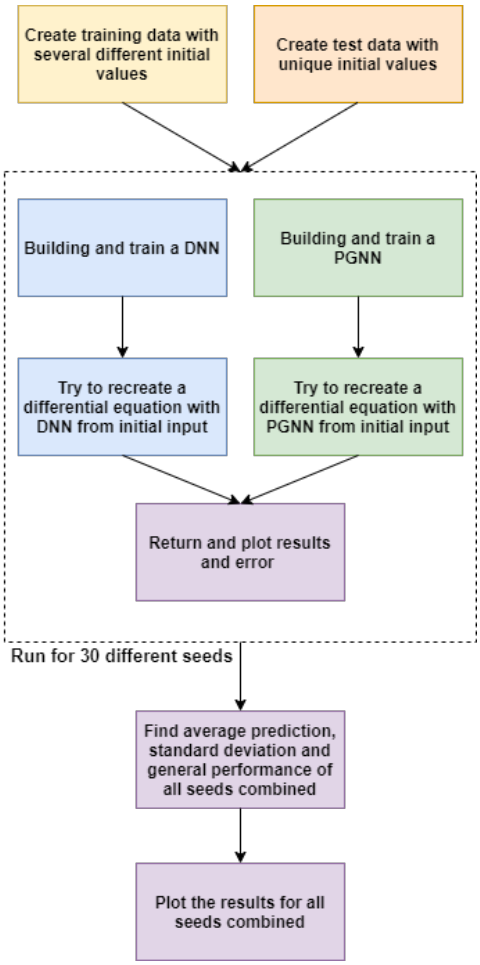


Figure 3.2.1: Workflow of the code.

values. All the runs with different seeds use the same data sets to train and test on.

3.2.3 Differential equations

Python SciPy function *odeint* was used for solving the differential equations over a given time interval. This function solves an initial value problem for a stiff or non-stiff system of first order(4). This function switches automatically between stiff and non-stiff system. For the this function to work with second order problems the differential equations, shown in equation 3.2.1, was first transformed into two differential equations of first order, shown in equation 3.2.2 and 3.2.3. The time-steps was created by the NumPy function *linespace*.

$$\ddot{x} = f(x)g(x) + h(x) \tag{3.2.1}$$

$$\dot{z_1} = z_2 \tag{3.2.2}$$

$$\dot{z_2} = f(x)g(x) + h(x) \tag{3.2.3}$$

The state values, x for the Duffing equation and R for the Rayleigh-Plesset equation, it's derivatives and double derivatives solved equation was then sorted into sets of x for input, y for reference, and $h(x)$ and $g(x)$ for the known dynamics, to be used in training and testing of the DNNs and the PGNNs.

3.2.4 Building and training the NNs

As stated in Section 3.1 TensorFlow was used to build the NNs. The *keras* packages were used to create the structure and layers of the networks. The networks were trained to output the double derivative for the current x (or R for Rayleigh-Plesset Equation) value. The *concatenate* function from *keras* was used for injecting the known dynamics in the PGNNs. The output for this part was trained models that was used to find the double derivative of the state in the testing. In addition, the training and validation loss was plotted and saved.

3.2.5 Testing the performance of the NNs

Separate test functions were created for testing the performance of the NNs in recreating a differential equation in a simulation setting. This was the reason that seeds where used instead of dropout to find uncertainty, as dropout would find the uncertainty in training and validation, but not show the uncertainty when the networks were used to do predictions in the simulation loop.

The test function started with a set of initial values, with the same dimension as the sets used to train the network. That mean that if the NNs were trained with x_t to $x_{(t-4)}$ the initial set would be the first 5 x values of the test set. These where then given as input to the differential equation, in the case of the PGNNs $g(x)$ and $h(x)$ for that time-step was also given as injection inputs, which returned the double derivative of the input. The output double derivative was then used to update the derivative, and next state value. The input then was updated for the next time step so x_t became $x_{(t-1)}$ and so on, and the new state value was added as x_t. For the PGNNs $g(x)$ and $h(x)$ were also updated. This was repeated for the length of the test set. Figure 3.2.2 illustrate the main idea for the test loop.

3.2.6 Plotting and saving results

The combined average prediction, error from reference, and standard deviation for all seeds were then calculated, saved, and plotted. The error was calculated

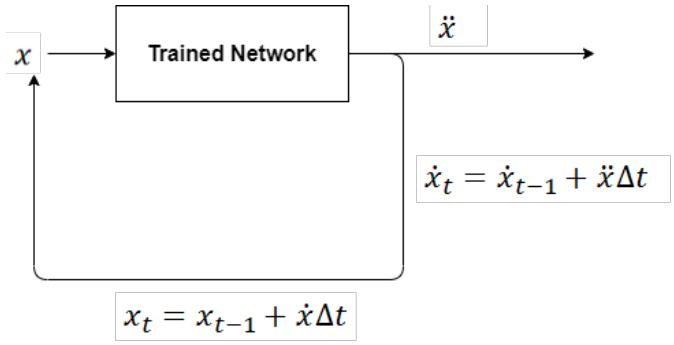


Figure 3.2.2: Test loop for the networks: This loops test how well the networks can recreate the solution to an ODE, by solving the double derivative and update the derivative, and state value based on this result.

as relative error which gave some problems when using a function oscillating around 0, which made these plots not very helpful in analyzing the results.

3.3 Unchanging parameter settings

There were done multiple tests with different sets of equations and parameter settings. Some parameters where kept constant in all tests, listed in table 3.3.1

Optimizer	Adam
Activation func.	Relu
Loss function	MSE
Validation split	0.2
Shuffle	True
Number of seeds	30

Table 3.3.1: The parameter settings that was kept constant for all tests.

A linear activation function was use instead of *Relu* for the single node output layer. Other parameters were tuned and changed depending on equation and performance of the networks.

3.4 Set-up for the differential equations

This section covers the differential equation used and their parameters. As well as initial values used in the data sets, and parameters used in the NNs for each equation.

3.4.1 Set-up for the Duffing equation

One of the equations that was used for testing was the Duffing equation. The Duffing equation is a oscillating non-linear second-order ODE that models damped and driven oscillators (9). The Duffing equation is shown in equation 3.4.1, where δ is the damping coefficient, α is the linear stiffness coefficient, β is the restoring force coefficient, γ is the amplitude of the periodic driving force, and ω is the angular frequency of the driving force. The values for each of these were kept constant for all test on the Duffing equation and are found in table 3.4.1.

$$\ddot{x} + \delta\dot{x} + \alpha x + \beta x^3 = \gamma \cos \omega t \tag{3.4.1}$$

δ	1
α	1
β	0.5
γ	3
ω	0.4

Table 3.4.1: The coefficients used for the Duffing equation.

The training and test sets where the same for all tests and seeds, with a time step length of 0.0025 and start at $t = 0$. The initial values for each of the generated data can be found in table 3.4.2.

Set	Initial x	Initial $\dot{x}$	End at $t =$
Training 1	1	0.5	20
Training 2	-2	0.8	10
Training 3	3	-1	15
Test	1.5	0.3	25

Table 3.4.2: The initial values used for each of the sets used for training and testing of networks on the Duffing equation.

The unchanged settings for all networks on the Duffing equation are listed in table 3.4.3.

Learning rate	0.00005
Early stop patience	40
Batch size	32

Table 3.4.3: Unchanged network settings for Duffing equation.

Equation 3.4.2 shows the $h(x)$ that was used, equation 3.4.3 shows the $g(x)$ that was used, equation 3.4.4 shows the y value that was used as output reference in training, and equation 3.4.5 shows the input set for each time step.

$$h(x) = \gamma \cos \omega t \tag{3.4.2}$$

$$g(x) = x_t^3 \tag{3.4.3}$$

$$y = \ddot{x}_t \tag{3.4.4}$$

$$X = \begin{bmatrix} x_t \\ x_{t-1} \\ x_{t-2} \\ x_{t-3} \\ x_{t-4} \\ x_{t-5} \\ x_{t-6} \\ t \end{bmatrix} \tag{3.4.5}$$

3.4.2 Set-up for the Rayleigh-Plesset equation

The Rayleigh-Plesset equation is a non-linear second-order ODE from fluid mechanics. The equation describes the dynamics of bubbles in in-compressible fluids (15). The Rayleigh-Plesset equation is shown in equation 3.4.6, where ρ_L is the density of the surrounding liquid, R is the radius of the bubble, ν_L is the kinematic viscosity of the bubble, and γ is the surface tension of the bubble-liquid interface. For the purpose of this project $\Delta P(t)$ was set to a exponential decaying function that started at 300 and converged to 0. The values for each of these were kept constant, aside from R, and had the values shown in table 3.4.4. These values are not meant to be realistic in this test, but that the values are chosen in a way that makes smooth results for the networks to train and test with, without any instabilities or singularities.

$$R\ddot{R} + \frac{3}{2}\dot{R}^2 + \frac{4\nu_L}{R}\dot{R} + \frac{2\gamma}{\rho_L R} + \frac{\Delta P(t)}{\rho_L} = 0 \tag{3.4.6}$$

ρ_L	997
ν_L	10^{-6}
γ	0.1
$\Delta P(T)$	300*(-e$^{\frac{0.1}{t+10^{-6}}}$)

Table 3.4.4: The coefficients and $\Delta P(t)$ equation used for the Rayleigh-Plesset equation. The 10^{-6} in $\Delta P(t)$ is to avoid instability at $t = 0$

Set	Initial R	Initial $\dot{R}$	End at $t =$
Training 1	3000	-7	2.9
Training 2	1500	1200	5
Training 3	100	600	1.5
Training 4	2000	300	2.5
Training 5	2500	0	2.5
Test	1800	400	3

Table 3.4.5: The initial values used for each of the sets used for training and testing of networks on the Rayleigh-Plesset equation.

All generated sets for training and testing started from $t = 0$ and had time step size of 0.001. The initial values for the sets can be found in table 3.4.5.

The NN settings that was kept constant for all networks are listed in table 3.4.6.

Learning rate	0.0000001
Early stop patience	100
Batch size	32

Table 3.4.6: Unchanged networks settings for Rayleigh-Plesset equation..

Equation 3.4.2 shows the $h(x)$ that was used, equation 3.4.3 shows the $g(x)$ that was used, and equation 3.4.4 shows the y value used as output reference in training. The X inputs in the networks for the Rayleigh-Plesset equation were R_t to R_{t-21}, and $\dot{R}_t$ to $\dot{R}_{t-21}$, and t.

$$h(x) = \frac{300 * (-e^{\frac{0.1}{t+10^{-11}}})}{\rho_L R_t} \tag{3.4.7}$$

$$g(x) = \dot{R_t}^2 / R_t \tag{3.4.8}$$

$$y = \ddot{R}_t \tag{3.4.9}$$

3.5 DNN set-up

Two different sets of DNNs were created for each of the differential equation to compare results with the PGNNs. One was to get a best possible result with many layers and no regard for run time. The other had some fewer layers to the point where the uncertainty got noticeable larger if one more layer was removed. For the Duffing equation two extra test with reduced DNNs were done.

3.5.1 DNN Set-up for the Duffing equation

For the Duffing equation there was one DNN with 14-layers and one with 18-layers. In addition one test was run for a 6-layered DNN and one for a 3-layered DNN with more epochs to compare with a PGNN with the same number of layers and epochs. These two are not shown here as they are the same architecture as their respective PGNN just without $g(x)$ and $h(x)$(see section 3.6).

14-Layers

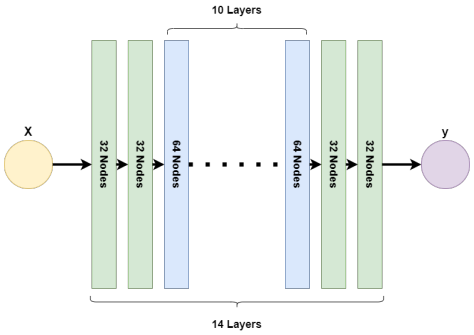


Figure 3.5.1: Architecture of the DNN with 14-layers for the Duffing equation.

18-Layers

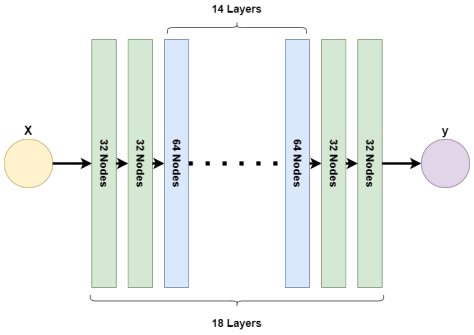


Figure 3.5.2: Architecture of the DNN with 18-layers for the Duffing equation.

3.5.2 DNNs for the Rayleigh-Plesset equation

The DNN with 36-layers was made early to create a reference to what a DNN would return with no regards to run time. Then it was reduced step by step until ending up on 10-layers. At less than 10-layers the DNNs created diverging results in the test loop.

10-Layers

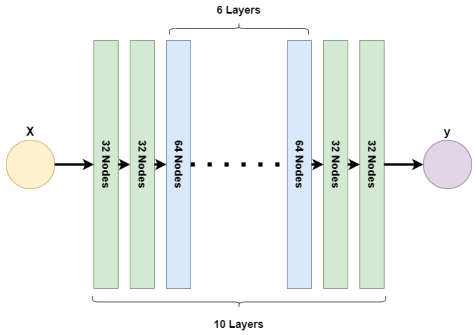


Figure 3.5.3: Architecture of the DNN with 10-layers for the Rayleigh-Plesset equation.

36-Layers

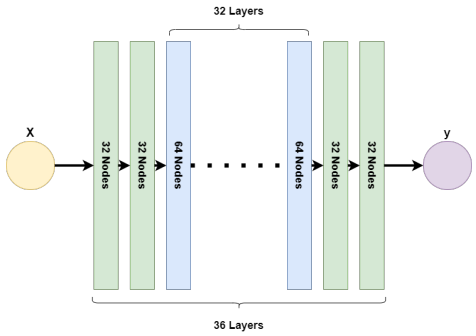


Figure 3.5.4: Architecture of the DNN with 36-layers for the Rayleigh-Plesset equation.

3.6 PGNN set-up

Several different architectures for PGNNs were created and tested. A PGNN set-up with 6-layers were created to be used for both equations.The different architectures used on this 6-layered set-up tested the effect on inserting $g(x)$, and the effect of the placement of the insertion. After these test were done the networks were reduced for each equation, and ended up with different number of layers, these reduced PGNNs are described in Section 3.7 and 3.8. Also, a couple of different test altering $g(x)$ and/or $h(x)$ were ran on the reduced network.

There were several reasons to test with a network with the same number of layers. One being to test generalizability of the network architecture, seeing

from results that this number of layers worked well for both equations, and saving workload not having to create a new architecture. Note however that some parameters in the networks, such as learning rate and epochs, had to be tuned for the specific equation, and used the parameter values listed in Section 3.4.

3.6.1 Only inserting $h(x)$

Here only $h(x)$ was given as extra input to the network, and it was given at the output layer. The position at the output layer was chosen because $h(x)$ contained all information, including coefficients, so it had no information that had to be trained for by the NNs. The architecture of this network is shown in figure 3.6.1.

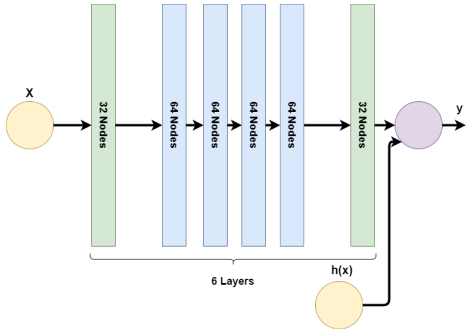


Figure 3.6.1: Architecture of the PGNN where only $h(x)$ was inserted at the output layer.

3.6.2 Inserting $g(x)$ early in the network

In this case $g(x)$ is inserted early in the network and goes through most layers to tests its effect on the network performance. In addition, $h(x)$ is given at the output layer. The architecture of this network is shown in figure 3.6.2.

3.6.3 Inserting $g(x)$ in the middle of the network

With this architecture $g(x)$ is inserted in the middle of the network and goes through half of the layers to tests effect of this placement on network performance. In addition, $h(x)$ is given at the output layer. The architecture of this network is shown in figure 3.6.3.

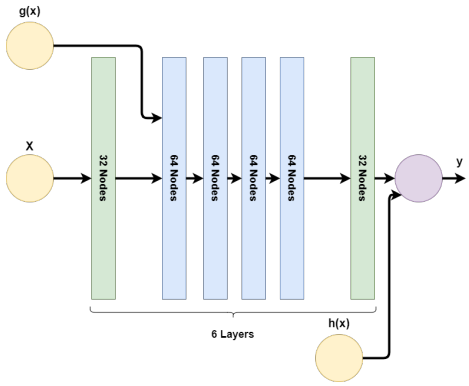


Figure 3.6.2: Architecture of the PGNN with $h(x)$ at the output layer and $g(x)$ inserted early in the network.

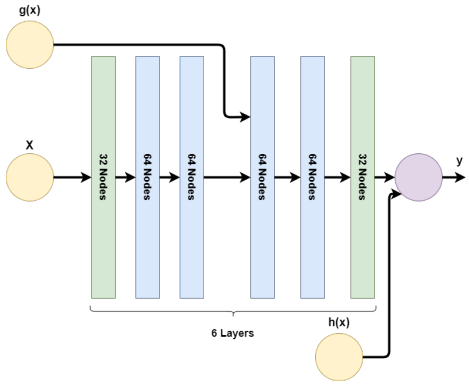


Figure 3.6.3: Architecture of the PGNN with $h(x)$ at the output layer and $g(x)$ inserted in the middle of the network.

3.6.4 Inserting $g(x)$ late in the network

The final shared PGNN has $g(x)$ inserted late in the network and goes through only one layer to tests the effect of late insertion of $g(x)$ on the network performance. In addition, $h(x)$ is given at the output layer. The architecture of this network is shown in figure 3.6.4.

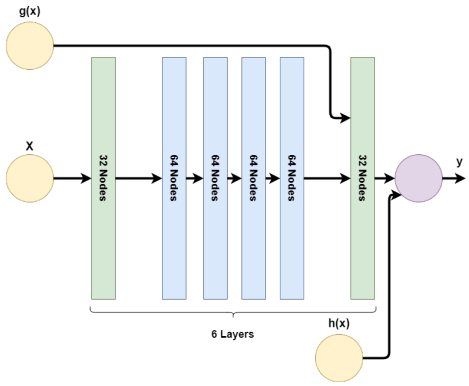


Figure 3.6.4: Architecture of the PGNN with $h(x)$ at the output layer and $g(x)$ inserted late in the network.

3.7 Reduced PGNN for the Duffing equation

Different insertion placements of $g(x)$ were again tested after a reduced PGNN was found for the Duffing Equation. The middle insertion placement of $g(x)$ was not tested as the reduced network did not have a natural difference between early or middle insertion unless $g(x)$ had been a part of the X input. It was also tested to bring $h(x)$ from the output layer to the layer before the output layer, as well as a test that removed the $h(x)$ input all together. The following reduced PGNN architectures were tested for the Duffing Equation:

3.7.1 Reduced PGNN with $g(x)$ inserted early in the network and $h(x)$ inserted at output layer

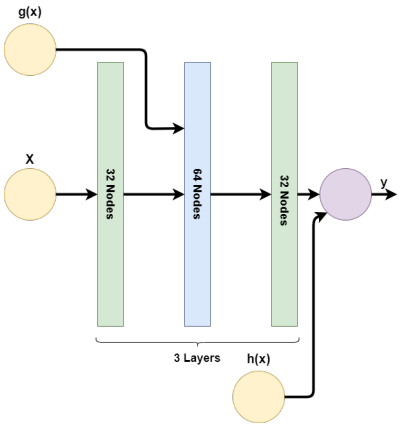


Figure 3.7.1: Architecture of the reduced PGNN with $h(x)$ at the output layer and $g(x)$ inserted early in the network for the Duffing equation.

3.7.2 Reduced PGNN with $g(x)$ inserted late in the network and $h(x)$ inserted at output layer.

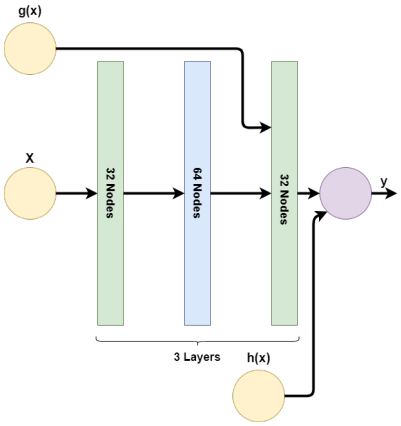


Figure 3.7.2: Architecture of the reduced PGNN with $h(x)$ at the output layer and $g(x)$ late in the network for the Duffing equation.

3.7.3 Reduced PGNN with $g(x)$ inserted early in the network and $h(x)$ Inserted Late in the Network

Here it was tested what effect it would have if $h(x)$ was included in a layer inside the structure of the PGNN instead of at the output layer. Here $g(x)$ was chosen to be inserted at a layer before $h(x)$.

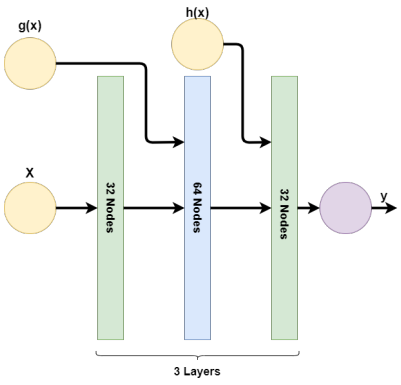


Figure 3.7.3: Architecture of the reduced PGNN with $h(x)$ inserted late in the network and $g(x)$ inserted early in the network for the Duffing equation.

3.7.4 Reduced PGNN with $g(x)$ inserted late in the network and no $h(x)$

This test was done to see which effect $h(x)$ had on the performance of the network. By doing this it is possible to do some analysis on how the different known dynamics influences the system.

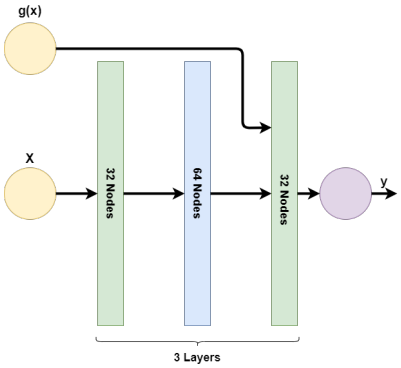


Figure 3.7.4: Architecture of the reduced PGNN with inserted late in the network and no $h(x)$ for the Duffing Equation.

3.8 Reduced PGNNs for the Rayleigh-Plesset Equation

For the reduced PGNN of Rayleigh-Plesset only one placement of $g(x)$ was tried because the reduced PGNN only had to layers, where $g(x)$ had to go between the layer to not be part of input or output layer directly. One test was done with different $h(x)$ to what effect the different $h(x)$ would have on the performance of the network.

3.8.1 Reduced PGNN with $g(x)$ and $h(x)$

Figure 3.8.1 showes the architecture of the reduced PGNN that was used on the Rayleigh-Plesset equation.

3.8.2 Reduced PGNN with $g(x)$ and Different $h(x)$

This test is design to test the effect of changing the $h(x)$ for the Rayleigh-Plesset equation in the 2-layered PGNN. The new $h(x)$ used here is given in equation 3.8.1.

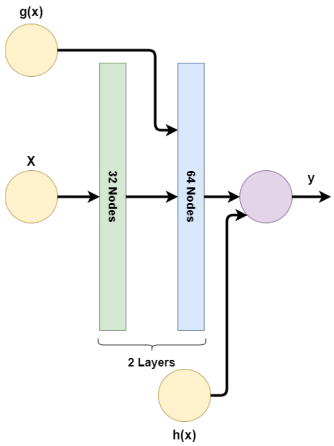


Figure 3.8.1: Architecture of the reduced PGNN with 2-layers for the Rayleigh-Plesset equation.

$$h(x) = \frac{-2\gamma}{\rho_l R^2} \tag{3.8.1}$$

Chapter 4

Results and Discussions

The first part of this section will present the results for all setups covered in Chapter 3.The results are given in two sections, one for each equation, making it easy to compare the results of the different network architectures. The results presented will be a used to discuss the use of PGML and possibilities it can create. These topics will be discussed in the later part of the chapter, in Section 4.3.

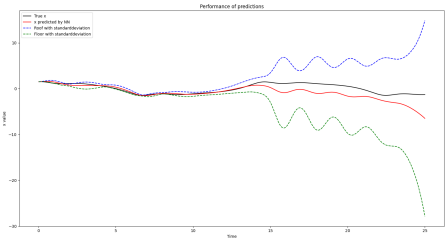
4.1 Results for the Duffing equation

This section presents the results of the simulation for predicting the Duffing Equation with the different network types described in Chapter 3. Some training and validation loss are also presented to discuss the performance of the networks. The results from the DNNs are first covered, followed by the results from the different PGNNs, and finally some test results with both type of networks with same number of layers are presented.

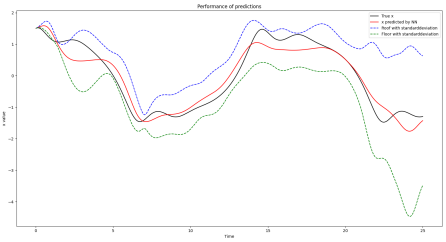
4.1.1 Results from the DNNs

With the DNN of 14-layers there are large error in the predictions for some of the seeds. This can be seen by the large standard deviation in figure 1a. As it gets close to the first top it starts to get oscillating error. This indicates that the network trains for an oscillating function, but for some seeds it struggles to follow the new initial value problem and starts to diverge from it. From looking at the loss during the training phase, figure 2a, it indicates that more than 150 epochs might give be needed.

The network with 18-layers, shown in figure 1b, performs better than the one with 14-layers, but still has large standard deviation, especially towards the end of the simulation. It seems that also in this network with 18-layers

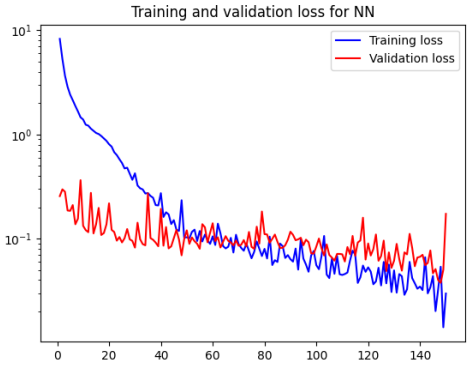


(1a) Simulation using 14-layered DNN.

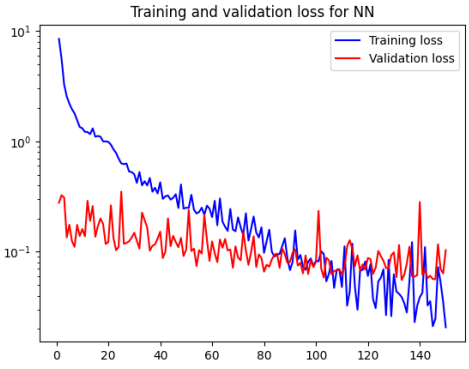


(1b) Simulation using 18-layered DNN.

Figure 4.1.1: Simulation performance for the different DNN architectures on the Duffing Equation



(2a) Loss using 14-layered DNN.



(2b) Loss from Duffing Equation using 18-layered DNN.

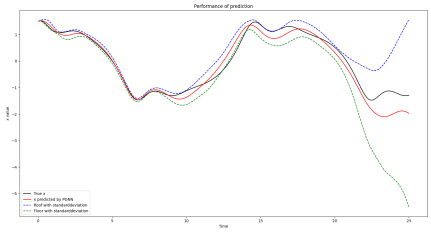
Figure 4.1.2: Loss function values for the different DNNs architectures from training on the Duffing Equation

needs more than 150, as seen by the decreasing training loss in figure 2b. However, the average prediction is following the shape of the reference, with some inaccuracies.

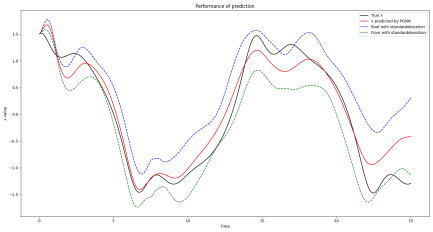
In both cases the results are not accurate over time, and which makes them untrustworthy to be used in systems that require accurate predictions. The loss function values for both networks are fluctuating and reach similar end values.

4.1.2 Results from the PGNNs

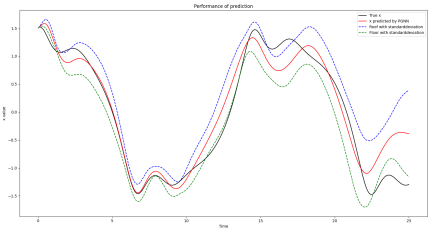
The results in figure 3a shows that with the introduction of $h(x)$ at the output layer of the network the performance in simulating the Duffing equation is accurate for the first half of the simulation, but has large standard deviation towards the end, with the simulation for some seeds diverging. Though the standard deviation is large the average prediction follows close to the reference.



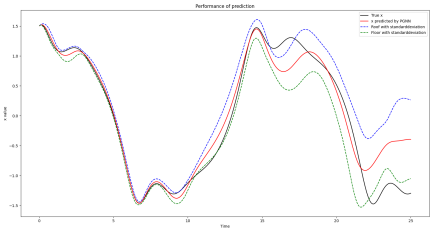
(3a) PGNN with only $h(x)$.



(3b) PGNN with early insertion of $g(x)$.



(3c) PGNN with insertion of $g(x)$ in the middle.



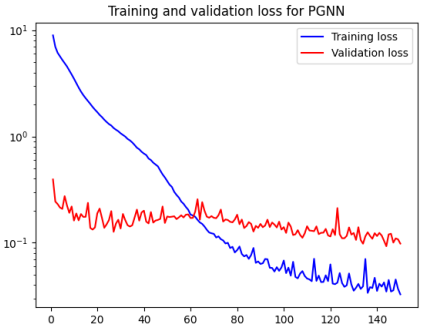
(3d) PGNN with late insertion of $g(x)$.

Figure 4.1.3: Simulation performance on the Duffing Equation with the different PGNN architectures with 6-layers: The $h(x)$ figure has different y-axis values as the standard deviation diverges.

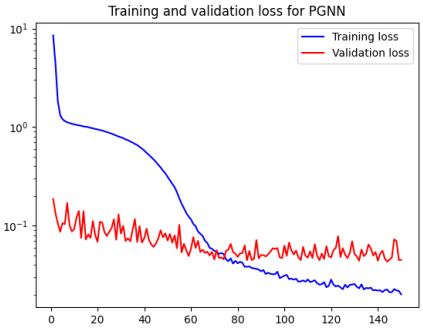
The average prediction for $h(x)$ follow the reference better than the predictions form the DNNs.

From the figures 3b, 3c and 3d it can be seen that adding the insertion of $g(x)$ gives better results with both average prediction and standard deviation, especially at the later stages. There is still room for improvements, and at the end the reference is outside of the standard deviation intervals for all three options. It also shows that for the give $g(x)$ that later insertion gives better accuracy and less standard deviation than early insertion for the Duffing equation from time 0 to 5 the simulation with $g(x)$ inserted early and in the middle actually has somewhat worse performance than the simulation with only $h(x)$.

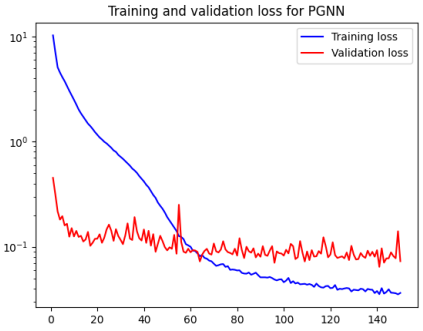
Figure 4.1.4 shows that the validation loss doesn't change much after the first 40 epochs for all the PGNNs, but the training loss has a strong declining curve for the first 80 epochs, that flattens out towards the final epochs. The training loss is close to smooth, aside for the one with only $h(x)$ from epoch number 80. The validation loss is fluctuating somewhat for all of the networks, but less than the fluctuation of validation loss for the DNNs.



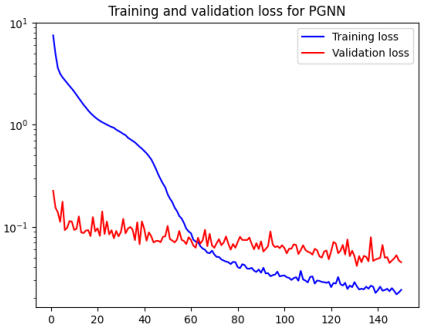
(4a) PGNN with only $h(x)$.



(4b) PGNN with early insertion of $g(x)$..



(4c) PGNN with insertion of $g(x)$ in the middle.



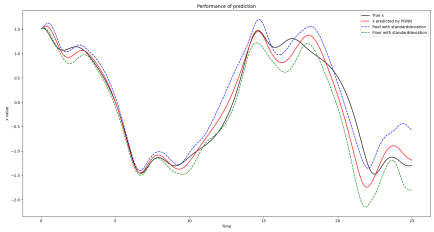
(4d) PGNN with late insertion of $g(x)$.

Figure 4.1.4: Loss function values for the different PGNN architectures with 6-layers on Duffing Equation.

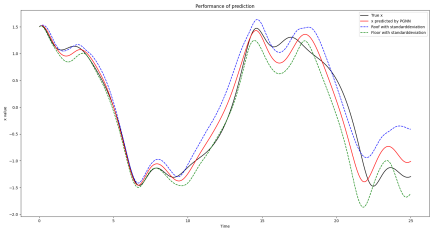
4.1.3 Results from the reduced PGNNs

For the test on the reduced networks the number of epochs were increased because of the training loss took more epochs to get smaller, and to get to the same level as validation loss. The loss from training on the reduced PGNNs are shown in figure 4.1.6. As seen in this figure the training loss curves are smooth and still declining at the end of epochs. This indicates that more epochs might have given even better results. However the one with the lowest training loss, where $h(x)$ is in the middle as shown in figure 6c, does not have the best prediction results in the differential equation simulation.

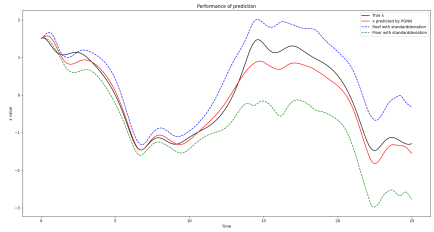
The results for early and late injection of $g(x)$ for the reduced (4-layers) PGNN, figures 5a and 5b, showed accurate predictions for most of the simulation with little standard deviation. There are still some deviations at the top of the oscillation and the second bottom, however. When $h(x)$ was changed to the middle of the network the uncertainty increased drastically, but the average prediction was still close to the reference. Since $h(x)$ was $\gamma \cos \omega t$ it was a known



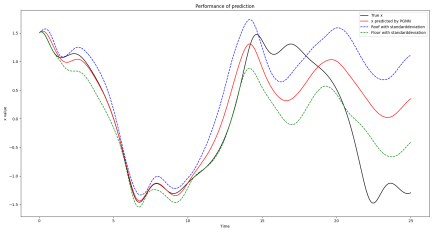
(5a) Reduced PGNN with early insertion of $g(x)$.



(5b) Reduced PGNN with late insertion of $g(x)$.



(5c) Reduced PGNN with late insertion of $g(x)$ and $h(x)$ in the middle.



(5d) Reduced PGNN with $g(x)$ inserted late, and no $h(x)$.

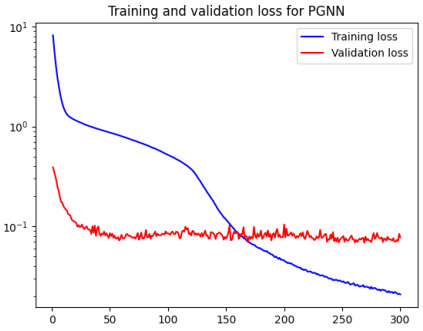
Figure 4.1.5: Simulation performance of the different PGNN architectures with 3-layers on Duffing Equation.

value with no uncertainty or dependencies on early predictions. This might be the reason this architecture performs worse than with $h(x)$ at the output layer, as weights gets trained for $h(x)$ where they should really be. The worst results however were when $h(x)$ was removed altogether. From the top point of the function in this simulation, figure 5d, the predictions starts to create a new oscillating function that does not match the reference function. It seems that the PGNN in this simulation struggles over time to keep predicting the same function due to uncertainty that gets carried over from previous predictions, which leads it to not following the shape of the reference.

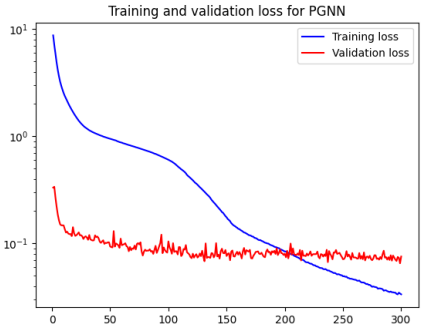
4.1.4 Comparison of DNNs and PGNNs with same number of layers

Figure 4.1.7 shows how DNNs and PGNNs with the same number of layers, and epoch (here 300) perform on predicting the Duffing equation. In the figure the filled areas are the standard deviation for its respective network (same color). As can be seen here the PGNNs has a lot less standard deviation and better accuracy. The difference between the performance of the DNN and the PGNN is especially clear when they only have 3-layers.

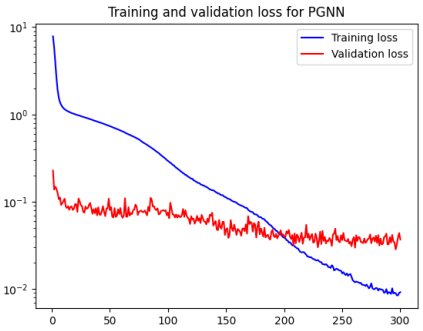
One interesting observation is that the 3-layered PGNN has less standard



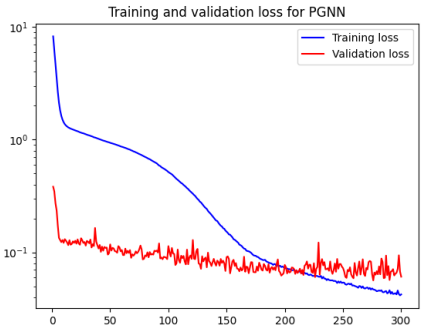
(6a) Reduced PGNN with early insertion of $g(x)$.



(6b) Reduced PGNN with late insertion of $g(x)$.



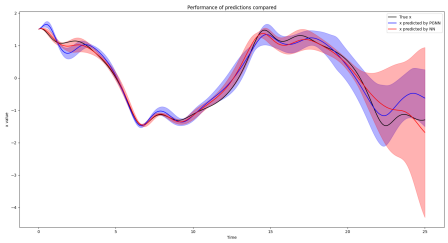
(6c) Reduced PGNN with late insertion of $g(x)$ and $h(x)$ in the middle.



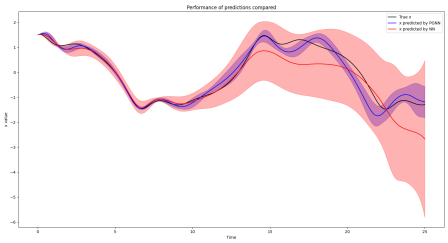
(6d) Reduced PGNN with only $g(x)$.

Figure 4.1.6: Loss function values for the different PGNN architectures with 3-layers on Duffing Equation.

deviation than the one with 6-layers, and that at some points, especially early in the simulation its average prediction is actually more accurate. Although at some points the reference is not covered within the standard deviation interval of the 3-layered PGNN, while this does not happen with the 6-layered PGNN.



(7a) Compared results of DNN and PGNN performance with 6 layers.



(7b) Compared results of DNN and PGNN performance with 3 layers.

Figure 4.1.7: Comparing the performance of PGNN and DNN at recreating the Duffing equation: Results from PGNN is shown in blue, DNN is shown in red, and true value is shown in black. The color fill is standard deviation for its respective network.

4.2 Results for the Rayleigh-Plesset equation

This section presents the results of the simulation for predicting the Rayleigh-Plesset equation with different network architectures. Some learning rate will also be presented to discuss the performance of the networks. The results from the DNNs are first covered, followed by the results from the different PGNN networks.

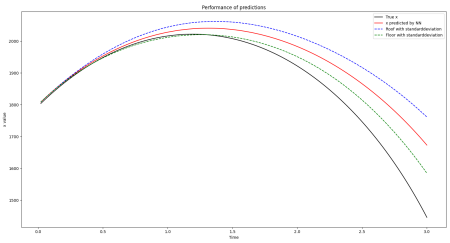
In general the predictions did not follow the same curve as the reference, the reasons for this is not found in this work, but will be discussed in Section 4.3.

4.2.1 Results from the DNNs

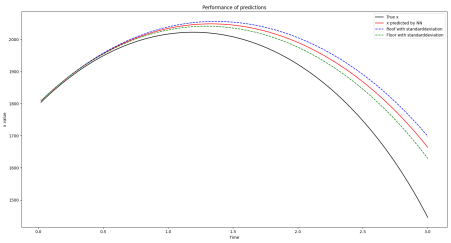
As can be seen in figure 4.2.1 the 10-layered DNN has some more standard deviation in the results than the 36-layered one, but the 10-layerd modeled used 500 instead of 700 epochs. On both test there are deviation from the reference curve, with both of the DNNs creating a curve with higher R values. From the training and validation loss, seen in figure 4.2.2, the networks never manage to get either of the loss values low, which indicates that the networks struggles to learn the function from the training data.

4.2.2 Results for the PGNNs

The same deviation from the reference curve that was present with DNNs is also present in all the test with the 6-layered PGNN, shown in figure 4.2.3. The interesting results from these tests are that having just $h(x)$, and having $h(x)$ and late $g(x)$ have very similar prediction and standard deviation. While insertion of $g(x)$ in the middle of the network or early in the network have

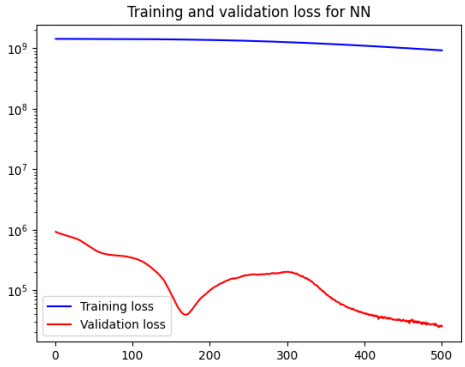


(1a) 10-layered DNN.

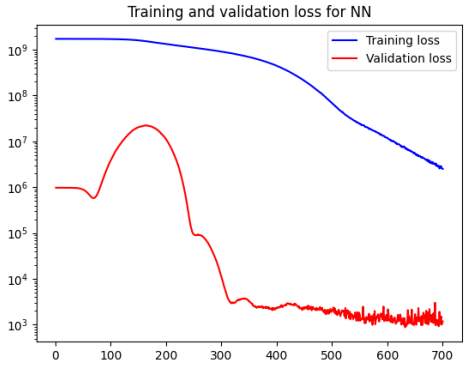


(1b) 36-layered DNN.

Figure 4.2.1: Simulation performance of the different DNN architectures on the Rayleigh-Plesset Equation.



(2a) 14-layered DNN.



(2b) 36-layered DNN.

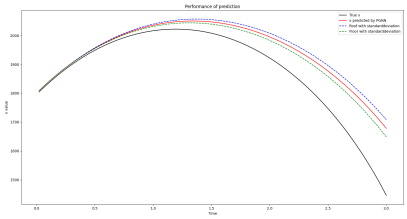
Figure 4.2.2: Loss functions values for the DNNs on the Rayleigh-Plesset Equation.

less standard deviation. Insertion of $g(x)$ early in the network removes almost all standard deviation, which indicates that for some ODEs early insertion of a $g(x)$ can remove variance from initial weight conditions, which are dependent on the seed.

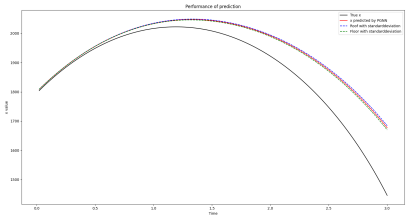
For the loss during training, seen in figure 4.2.4 the PGNNs had better results than the DNNs, but they still struggled to get low loss values, and training loss never reaches the validation loss. More epochs might give better result, as the training loss is still decreasing at the final epoch.

4.2.3 Results for the reduced PGNNs

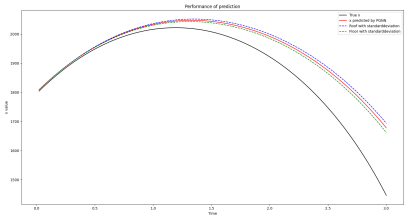
With the reduced PGNNs the results have more standard deviation, seen in figure 4.2.5 than for the 6-layered PGNNs. Although the average predictions are still similar to the other networks. For the loss functions in figure 4.2.6 the training loss never properly starts to decline. It is indicated that more epochs might be need from the training and validation loss. However, it was tested to



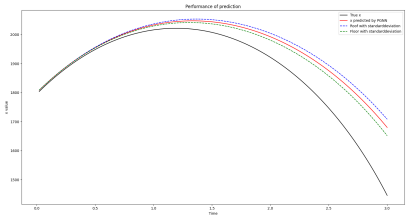
(3a) PGNN with only $h(x)$.



(3b) PGNN with early insertion of $g(x)$.



(3c) PGNN with insertion of $g(x)$ in the middle.

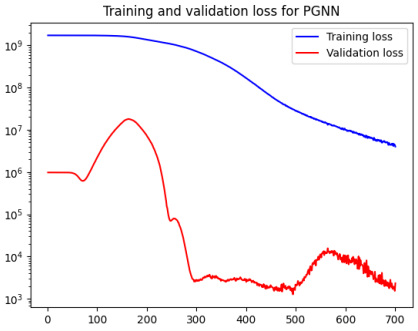


(3d) PGNN with late insertion of $g(x)$.

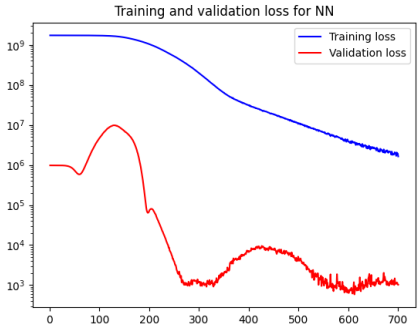
Figure 4.2.3: Simulation performance of the PGNN architectures with 6-layers on the Rayleigh-Plesset.

increase number of epochs drastically, but this only made the simulation have even larger deviation from the reference.

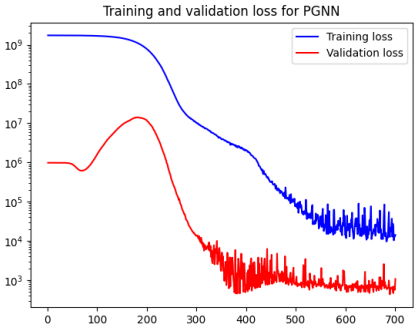
One interesting observation is that the two different $h(x)$ used in the test for the reduced PGNN here does not impact the results noticeably. This indicates the the $h(x)$ that was chosen does not have a large impact on the network in general, which is differs from what was found on from the tests on the Duffing equation seen in section 4.1. This indicates that the properties of $h(x)$, as well as how much it dominates the ODE, has a large impact on how much potential it has to improve a network.



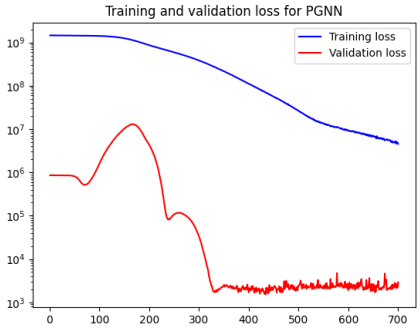
(4a) PGNN with only $h(x)$.



(4b) PGNN with early insertion of $g(x)$..

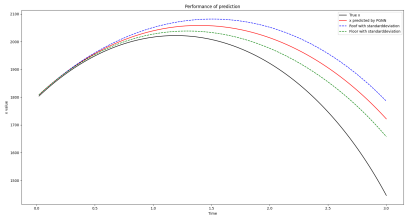


(4c) PGNN with $g(x)$ inserted in the middle.

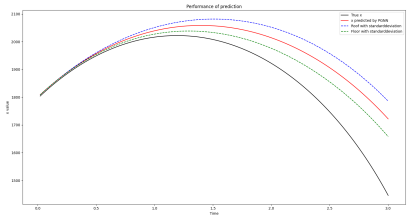


(4d) PGNN with late insertion of $g(x)$.

Figure 4.2.4: Loss function values for the PGNN architectures with 6-layers on the Rayleigh-Plesset Equation.

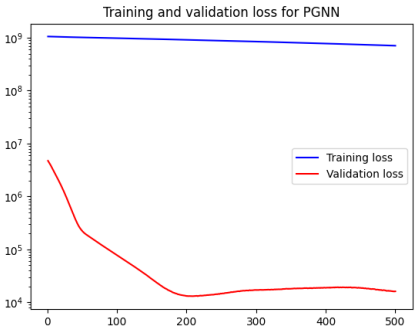


(5a) Reduced PGNN.

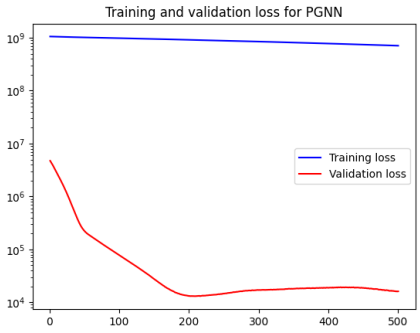


(5b) Reduced PGNN with different $h(x)$.

Figure 4.2.5: Simulation performance of the reduced PGNN architectures with 2-layers on the Rayleigh-Plesset Equation.



(6a) Reduced PGNN.



(6b) Reduced PGNN with different $h(x)$.

Figure 4.2.6: Loss function values for the reduced PGNN architectures with 2-layers on the Rayleigh-Plesset Equation.

4.3 Discussion

This section will discuss the results, other similar work on PGML, what kind of algorithms PGML are suited to be used in, and how PGML can create value for different fields, such as engineering and economics.

4.3.1 Discussing the results

In context of the motivation and research question there are some parts of the results that are worth discussing. Firstly, the differences in performance, both in accuracy and uncertainty, between DNNs and PGNNs, as a major motivation for this thesis was to study if PGNNs can outperform DNNs, also in reduced forms for ODEs. Furthermore, it is interesting to compare the results for different architectures of PGNN and performance on different functions. From the results there are some differences in performance based on architecture, and for all networks performance was better on the Duffing equation than the Rayleigh-Plesset equation.

Performance differences between DNNs and PGNNs

From the results for the Duffing equation, it is clear that the PGNNs performs better than then DNNs in both accuracy and standard deviation. This is becomes especially clear in figure 4.1.7 where DNNs and PGNNs with same number of layers are compared. A reduced version of the PGNN with only 3-layers outperforms 3-layered, 6-layered, 14-layered and 18-layered DNNs. This means that the extra input given from known physics drastically improve performance in the case of the Duffing equation, especially the insertion of $h(x)$. However, the same conclusion cannot be drawn from the results for the Rayleigh-Plesset equation. Both the DNNs and PGNNs did not performed well in the simulations for the Rayleigh-Plesset equation. Although the PGNNs had in general less standard deviation than the DNNs, which indicates that the extra information about the dynamics remove some of the variation that comes from the weight's initial conditions.

Performance of the different PGNN architectures

From the Duffing equation the results shows that all PGNN architectures that included both $g(x)$ and $h(x)$ performed well. The 3-layered and 6-layered networks had different strengths and weaknesses, where the 3-layered one had less standard deviation, but also did not always have the real value inside the standard deviation bounds. On the other hand, the 6-layered one has somewhat more standard deviation, but the real value was always in range of the simulations standard deviation bound. The reason for the 3-layered one having

less standard deviation might be that with less nodes there are less difference between seeds, as the amount of initial weight that gets changed are less, since there are less weights with 3-layers than 6-layers.

The location of the insertion of $g(x)$ did not have significant effect on the results for the Duffing Equation, but the test without $g(x)$ shows the simulation diverging for some seeds towards the end. Changing the location of $h(x)$ on the other hand had major implications on the performance. By inserting $h(x)$ in the middle of the network instead of the end increased the standard deviation, and completely removing $h(x)$ made the simulation prediction deviate from the reference and simulating a oscillating function that did not follow the shape of the reference, and thus had high error.

For the Rayleigh-Plesset equation changes to $g(x)$ have a clearer effect. By having $g(x)$ early the standard deviation based on seeds are drastically decreased. Furthermore, the effect of changing $h(x)$ in the simulations on the Rayleigh-Plesset equation were small, which indicates that the $h(x)$ chosen for the Rayleigh-Plesset equation did not have enough information to have large impact on the performance of networks.

All this indicates that the performance of the PGNNs are very dependent on the properties of the function it is predicting, $g(x)$, and $h(x)$. From the Duffing equation is seems that adding in oscillating functions, such as $cosine$ or $sine$, will improve the network drastically. Also, from the results it shows that different amount of layer has different advantages and disadvantages when it comes to accuracy and standard deviation. In general, the training loss for fewer layers has a somewhat slower decline and might require more epochs, but in return it can in certain instances remove some of the variation that comes from seeding.

The deviation in simulations for the Rayleigh-Plesset equation

For all networks tested there were some deviations between the reference and network simulated curve for the Rayleigh-Plesset equation. As seen from the loss plots for these tests the networks never quite managed to learn the input to output properly. There can be several reasons for this:

- Training data might be inadequate
- Learning rate might be non-optimal
- Input to networks might not be enough to learn the full function
- Architectures of the networks might not be optimal for the Rayleigh-Plesset Equation

Several of these points where somewhat tested, but there was not set aside time to test them properly and structurally to find the exact reason for the

deviation and resolve it. This is something that has to be worked more on to use PGNNs efficiently on the Rayleigh-Plesset equation.

4.3.2 Similar work

From research on the topic only two other instances of PGML being used was found. One uses PGML on calculating lift and drag coefficients on airfoils (30), and the other uses PGML in long-shot term memory network with the Galerkin projection model(33). Both these papers shows that PGML can be used to make more generalizable network models and give decrease uncertainty in predictions. The results from these matches well with the results for the Duffing equation found in this thesis, as well as the decreased standard deviation for both of the ODEs. The uncertainty is reduced, the PGML approach handles better new initial conditions than the DNNs, and the PGNNs can be reduced in layers, thus being more appropriate to be used in online functionalities. From the results and other work there are strong indications that the PGML framework can be used to create NNs that are more explainable, generalizable, and more efficient than conventional DNNs.

Use of PGML to calculate wind-coefficient in windmill airfoils.

In this paper it is shown that the use of PGML works well in predicting the lift coefficients for airfoils. The networks with PGML especially thrive in the areas where the simplified physical models are valid, which is at small angles of attack.

Use of PGML in Model Fusion

This paper uses PGML to run faster simulations of vortexes than the full physical equation would. By using simplified physical models and introducing them into the network it creates networks that solves the problem faster than a solver for the full physical system. This opens up opportunities in simulations and digital twins. The results from this paper shows that PGML handles oscillations better than DNNS, which is also shown with the results from the simulation on the Duffing equation.

4.3.3 Use of PGML

PGML can be used to create more accurate networks with less layers than DNNs, as shown by the results, primarily on the Duffing equation, and the other work done on PGML. This opens up possibilities where PGML can efficiently be used and create good results. Two of these possibilities are to use PGML to use PGML on real life system with data from measurements and some known dynamics,

the other possibility that will be discussed here is to use PGML together with ROMs to create models that can be used in digital twins or other efficiency dependent applications.

Use PGML to do predictions of systems with partially known dynamics

Some complicated systems consist of some known dynamics, and some unknown dynamics that affects it. The known dynamics can be from physical-based modeling, but there is also possibilities to use dynamics from explainable data-driven modeling approaches. The PGNNs does still use some layers with little explainability, but it gives an accurate prediction with as much explainability as possible. This makes it easier to construct meta-models to explain the layers in the network. This can be used in several fields where there is a lot of measured data, not just in the engineering field, more on this in section 4.3.4.

Using PGML in combination with ROMs to get efficient simulators

The known physics information that is inserted into a PGNN can be on the form of ROMs. Using this combination one can simplify advanced solvers for complex equations that uses high amount of processing resources and run-time to simulate a system, as shown from the results on the Duffing equation, and in (30) and (33). As well as having the advantage of being able to handle non-modeled disturbances or factors that act on and have a impact on the system. As mentioned in these papers, as well the results from this thesis, PGML can be a powerful tool to be used in online monitoring and simulations.

4.3.4 Possible value creation of PGML

There are multiple areas where PGML can create value, based the results of this thesis, and the other two papers on this subject. With more explainablility from PGNNs than DNNs, and the reduced number of layers it opens up possibilities in many fields as this can give accurate, explainable and fast predictors and simulators.

Digital Twins

Using the PGML framework together with ROMs can be used in digital twins as digital twins relay on fast and accurate calculations, with a lot of different factors, some that does not have well defined models(32). When using the PGML framework in this setting it can, together with ROM or other simplified models, replace more complex differential equations solvers, both for ODE and PDE, and thus increase the efficiency of the calculations. This makes it possible for the digital twin to calculate more factors and use less computing resources

while still being accurate enough to be a digital representation of a system. In addition, it is also more suited to handled factors that are not well modeled.

Engineering

In real world applications of engineering for dynamic systems there is often unknown or not modeled parameters and disturbances. Some of these can be filtered out as noise, but some of these might have an impact on prediction and model accuracy. PGML can be used to model the system and creating the right controller, or other engineered systems, to create the best possible solutions. In these cases PGML can be described as a system for the combination of all the steps of modeling to create a hybrid model of the system, as described for BDC in Chapter 2.

Macro-Economics

As some research on of macro-economics are starting to use dynamic models, such as (8), there is a possibility to use these models to specific problems/research cases, together with PGML to get a better prediction and understanding of the dynamics and effects in macro-economics. If used correctly this could help different type of institutes to understand the dynamics of the given macro-economic system as well as do good economic decisions and create more value for society.

Micro-Economics

One example for the use of PGML in micro-economics challenges is prediction of the sales of seasonal products. A store can sell some seasonal product that they know has a general absolute value of $sine$ shape depending on week in the season, and some other factors that makes it not a perfect $sine$. By first finding the general $sine$ behavior, and then using PGML with the other measured factors it could be possible to get better predictions of weekly sales of the given product. This can increase the efficiency of the store, and make sure it has enough product in stock, while not ordering too much which could result in a sale to empty the storage.

Epidemiology

In 2020 there were a lot of different models predicting the spread of COVID-19 and possible effects of restrictions. Several of these models uses the SIR-model for spread of diseases(3). There was a lot of uncertainty and inaccuracies with these models, and their predictions had large variation depending on the source of the model. There could be possibilities to use the PGML framework together

with data from COVID-19 and other epidemics to create models that could give better predictions in the event of a new pandemic or epidemic, that can take the SIR model, or other models for the spread of diseases, and better combine it with the collected data. However further research is required to find how well PGML would perform at this task, and what kind of value it could create.

Chapter 5

Conclusion and future work

5.1 Conclusions

The major conclusions of the thesis are:

- PGNNs can give better results than DNNs. As shown with the results on the Duffing equation.
- A PGNN can make accurate predictions with fewer layers than a DNN. As shown with the comparison between PGNNs and DNNs with the same number of layers.
- The insertion placement of $g(x)$ can influence the results of the PGNN, and that the effect of where $g(x)$ is placed is dependent on the system it is simulating. For the Rayleigh-Plesset the early insertion of $g(x)$ gave less standard deviation, while for the Duffing equation the early insertion of $g(x)$ increased standard deviation for some intervals.
- The results, and other work, indicate that the $h(x)$ or $g(x)$ that improve a PGNN the most are oscillating functions, such as cos. This was clearly shown by in the test on the Duffing equation with the performance differences between when the $h(x)$, which includes cos, was inserted versus when it was not inserted.

In doing so we answer all the research questions mentioned in 1.3.2 thereby realizing all the secondary and primary objectives

5.2 Future Work

There are still more research questions that can be explored, both on the networks and equations used in this thesis, as well as other subjects where PGML

can be used, as mentioned in the discussion part. From the results and discussion the following research questions are left unanswered:

- What is the reason for the deviation between prediction and reference for the simulations on the Rayleigh-Plesset equation?
- What type of oscillating functions has a large impact on PGNN performance, and why do these have this impact?
- What other ODEs can use PGNNs for efficient and accurate predictions?
- How can PGML be combined with explainable data-driven models where no physical-based models are available?
- What is the possible value creation for PGML and which problems can they be used for in fields such as medicine, economics and epidemiology?
- How would PGNN perform with online training capabilities?

Several of these research questions can be one or multiple theses by themselves, and some of these questions are currently being researched by the Institute for Engineering Cybernetics.

Bibliography

[1] About python. https://www.python.org/about/ Accessed 10-June-2021.

[2] github repository for pgml aerodynamic forces prediction. https://github.com/surajp92/PGML Accessed 5-April-2021.

[3] Maa: Sir-model. https://www.maa.org/press/periodicals/loci/joma/the-sir-model-for-spread-of-disease-the-differential-equation-model Accessed 17-June-2021.

[4] odeint documentation. https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html Accessed 13-March-2021.

[5] Why tensorflow. ://www.tensorflow.org/about Accessed 10-June-2021.

[6] ACHDOU, Y., BUREA, F. J., LASRY, J.-M., LIONS, P.-L., AND MOLL, B. Partial differential equation models in macroeconomics. *Philosophical Transaction of Royal Society a Mathematical, Physical and Engineering Science* (2014).

[7] ALBAWI, S., MOHAMMAD, T. A., AND AL-ZAWI, S. Understanding of a convolutional neural network. *International Conference on Engineering and Technology (ICET)* (2017).

[8] ANDRESEN, T. On the dynamics of money circulation, creation and debt – a control systems approach. *Thesis for the degree of Doctor Philosophiae, NTNU* (2018).

[9] BRENNAN, M. J. *The Duffing Equation*. John Wiley Sons, 2011.

[10] CAPRA, M., BUSSOLINO, B., MARCHISIO, A., MASERA, G., MARTINA, M., AND SHAFIQUE, M. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. *IEEE Access Volume 8* (2020).

[11] CARUANA, R., AND NICULESCU-MIZIL, A. An empirical comparison of supervised learning algorithms. *ICML '06: Proceedings of the 23rd international conference on Machine learning* (2006).

[12] CHENG, Y., WANG, D., ZHOU, P., AND ZHANG, T. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine* (2018).

[13] CHOI, D., SHALLUE, C. J., NADO, Z., LEE, J., MADDISON, C. J., AND DAHL, G. E. On empirical comparisons of optimizers for deep learning. *arXiv:1910.05446v3* (2020).

[14] FATICHI, S., VIVONI, E., OGDEN, F. L., AND IVANOV, V. Y. An overview of current applications, challenges, and future trends in distributed process-based models in hydrology. *Journal of Hydrology* (2016).

[15] FRANC, J.-P. The rayleigh-plesset equation: a simple and powerful tool to understand various aspects of cavitation. *CISM International Centre for Mechanical Sciences* (2007).

[16] GOOGLE. Google ai experiments, 2021. `https://experiments.withgoogle.com/collection/ai` Accessed 12-Mai-2021.

[17] HOLZINGER, A., KIESEBERG, P., WEIPPL, E., AND TJOA, A. M. Current advances, trends and challenges of machine learning and knowledge extraction: From machine learning to explainable ai. *International Cross-Domain Conference for Machine Learning and Knowledge Extraction* (2018).

[18] HU, J., SASAKAWA, T., HIRASAWA, K., AND ZHENG, H. A hierarchical learning system incorporating with supervised, unsupervised and reinforcement learning. *Advances in Neural Networks – ISNN 2007* (2007).

[19] JORDAN, M., AND MITCHELL, T. Machine learning: Trends, perspectives, and prospects. *Science vol. 349* (2015).

[20] JOSHI, A. V. *Machine Learning and Artificial Intelligence*. Springer, 2020.

[21] KHANDELWAL, R. How to solve randomness in an artificial neural network? *Toward Data Science* (2020).

[22] LAPUSCHKIN, S., BINDER, A., AND MÜLLER, K.-R. Analyzing classifiers: Fisher vectors and deep neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[23] LAURET, P., FOCK, E., RANDRIANAIVONY, R. N., AND MANICOM-RAMSAMY, J.-F. Bayesian neural network approach to short time load forecasting. *Energy Conversion and Management Volume 49* (2008).

[24] LEVERMORE, C. D., AND OLIVER, M. Analyticity of solutions for a generalized euler equation. *Journal of Differential Equations Volume 133* (1997).

[25] MARTENS, H. Big data cybernetcis, 2016. `https://folk.ntnu.no/martens/?BigDataCybernetics` Accessed 5-June-2021.

[26] MONTAVON, G. Gradient-based vs. propagation-based explanations: An axiomatic comparison. *Lecture Notes in Computer Science book series (LNCS, volume 11700)* (2019).

[27] MÜLLER, K.-R., AND SAMEK, W. Towards explainable artificial intelligence. *Lecture Notes in Computer Science book series (LNCS, volume 11700)* (2019).

[28] NAGY, G. Ordinary differential equations, 2021. `https://users.math.msu.edu/users/gnagy/teaching/ode.pdf` Accessed 5-June-2021.

[29] O'LEARY, D. E. Artificial intelligence and big data. *IEEE Intelligent Systems (Volume: 28, Issue: 2)* (2013).

[30] PAWAT, S., RASHEED, A., SAN, O., AKSOYLU, B., AND KVAMSDAL, T. Physics guided machine learning using simplified theories. *Physics of Fluids volume 33* (2021).

[31] RAISSI, M., PERDIKARIS, P., AND KARNIADAKIS, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics Volume 378* (2019).

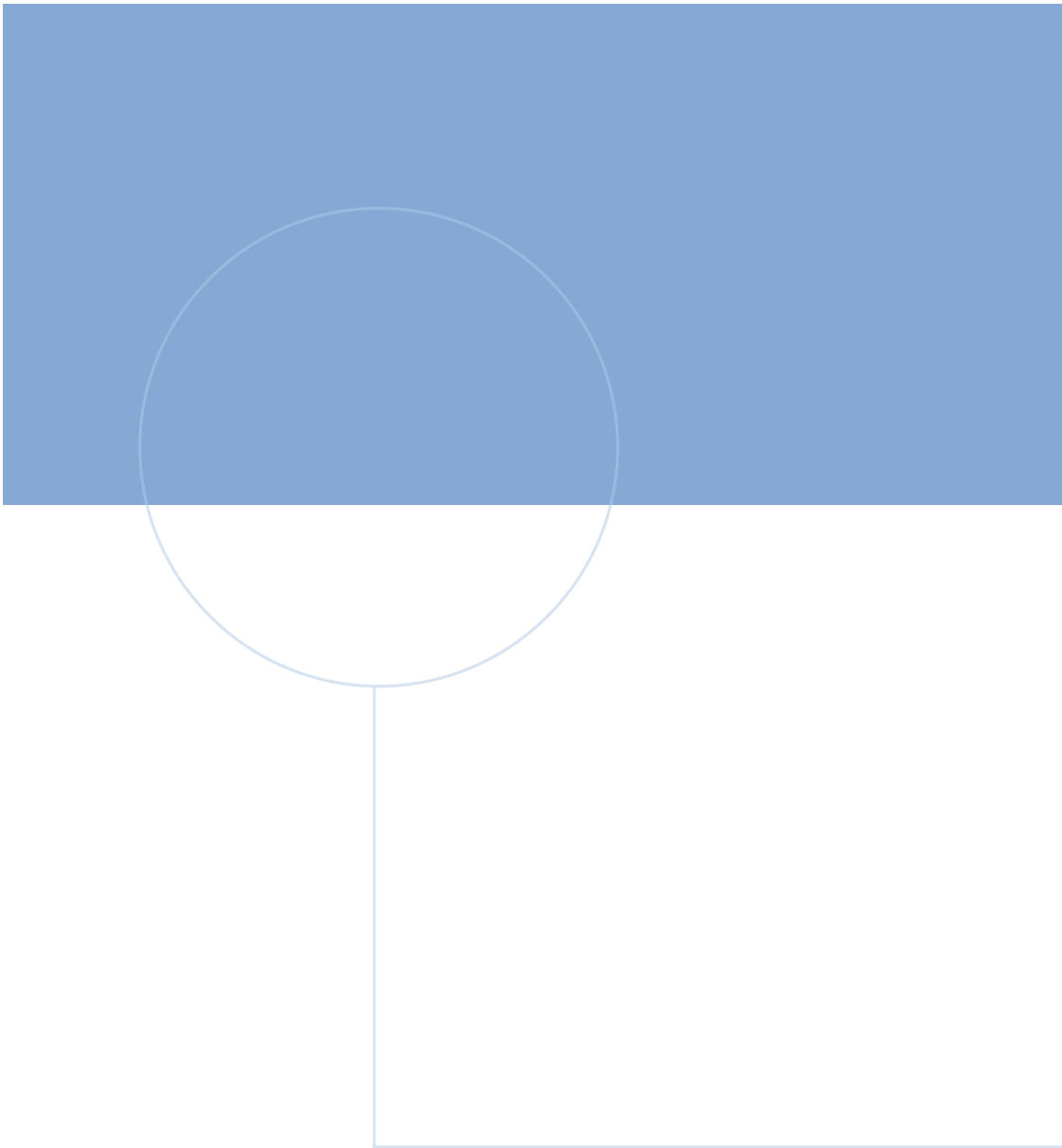
[32] RASHEED, A., SAN, O., AND KVAMSDAL, T. Digital twin: Values, challenges and enablers from a modeling perspective. *IEEE Access* (2020).

[33] RASHEED, A., SAN, O., PAWAR, S., NAIR, A., AND KVAMSDAL, T. Model fusion with physics-guided machine learning. *arXiv:2104.04574* (2021).

[34] RASHEED, A., SAN, O., ROBINSON, H., AND KVAMSDAL, T. Hybrid analysis and modeling as an enabler for big data cybernetics. *32nd Nordic Seminar on Computational Mechanics* (2019).

[35] SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development* (1959).

[36] SHADIEV, R., HWANG, W.-Y., CHEN, N.-S., AND HUANG, Y.-M. Review of speech-to-text recognition technology for enhancing learning. *Educational Technology Society vol. 17* (2014).



NTNU

Norwegian University of
Science and Technology