UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

INFORMED, INTERACTIVE, AND INTERPRETABLE MACHINE LEARNING FOR FORWARD KINEMATICS OF ROBOT ARMS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

BY

SAI TEJA KANNEGANTI
Norman, Oklahoma
2022

INFORMED, INTERACTIVE, AND INTERPRETABLE MACHINE LEARNING FOR FORWARD KINEMATICS OF ROBOT ARMS

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY THE COMMITTEE CONSISTING OF

Acknowledgements

Firstly, I would like to express my sincere gratitude to both of my advisors Dr. Dean Hougen and Dr. Jin-Song Pei for their continuous support, encouragement, excellent supervision, patience, tremendous knowledge, and making my doctoral research more enjoyable. I would like to thank both of them for being very supportive during my tough times. I enjoyed the broad technical discussions with Dr. Hougen which played a significant role in my job interviews. I would also like to thank Dr. Pei for the quick feedbacks she has provided which played an important role in timely graduation. I could not have asked for better advisors.

I would like to thank Dr. Sridhar Radhakrishnan for his guidance and trust in my abilities in the last six years. I would also like to thank Dr. Christan Grant and Dr. Charles Nicholson for their insights in improving this dissertation. Special thanks to Dr. Phillip Chilson for encouraging me to pursue Doctor of Philosophy.

I would like to thank my family (parents, grand parents, and brother) for supporting, motivating me through out this journey. Special thanks to my brother, Gowtham Teja Kanneganti for caring and sharing intellectual thoughts that helped me during this journry.

I would also like to thank CS graduate students and friends for making this journey more enjoyable. I would also like to thank CS non-teaching staff for their administrative support. I would like to thank REAL lab (Robotics, Evolution, Adaptation, and Learning Laboratory) for the technical discussions, opportunities, and collaborations it has provided me. Finally, I would like to thank the Gallogly College of Engineering and the School of Computer Science for the opportunities it has given me and for shaping me into what I am today.

Abstract

Machine learning (ML) is becoming increasingly sought after in diverse domains. Unfortunately for this objective, most ML research has focused too much on improving performance on evaluation metrics such as accuracy to the exclusion of other qualities like interpretability. However, to make important decisions, ML models need to be interpretable. The goal of interpretable machine learning (IML) is to build models that are understandable to users. One approach to IML is to have meaning to each of its components. Thus, IML aids in building models that are trustworthy and improve fairness in artificial intelligence. In informed ML, prior knowledge is explicitly integrated into the ML pipeline/training process. Interactive ML enables ML models to be interactively steered by people and is more advantageous for the tasks where human knowledge is needed in the analysis process. In this work, we proposed the I^3 framework that brings together the ideas of being informed, interactive, and interpretable. In this work we reintroduced, highlighted, and established the larger picture to one approach in the context of being informative, interactive, and interpretable. Pei et al.'s work is a strong candidate and is one instantiation of I^3 framework. In this work, Pei et al.'s work is used to approximate the kinematics of a robotic arm using interpretable artificial neural networks (ANNs). Pei et al.'s work is developed using applied mathematics for engineering mechanics and is based on approximating nonlinear functions where domain knowledge and visually observable features of the data are used to design ANNs. Pei et al.'s work is informed as scientific knowledge through applied mathematics, engineering and world knowledge through vision are represented in the form of algebraic equations, logic rules, and human feedback. The represented knowledge helps to narrow down the hypotheses for network architecture. Pei et al.'s work involve integrating prior knowledge obtained by examining the dominant features of the data. Then the interactive process involves choosing an appropriate basis function from the visualization of the function to be approximated; this helps in designing the ANN architecture and its initial values. Interpretability is the result of being informed and interactive. After analyzing Pei et al.'s work, we present a feasibility study approximating the kinematics of a simplified robotic arm. We extend Pei et al.'s work and its use for a different application, noting the challenges that arise while extending this work to more inputs and to multiple hidden layers. This approach leads to training success, good generalization, and interpretability.

Contents

Acknowledgements **iii**

Abstract **vi**

List of Figures **x**

List of Tables **xiii**

1 Introduction **1**
1.1 Introduction to interpretable machine learning 1
1.1.1 Informative 4
1.1.2 Interactive 4
1.1.3 Interpretable 4
1.2 Introduction to robot kinematics and robotic arm used 5
1.3 Intended contributions 6

2 Related Work **8**
2.1 Background of informed, interactive, and interpretable ML 8
2.1.1 Informed ML 8
2.1.2 Interactive ML 11
2.1.3 Interpretable ML 12
2.2 Fundamentals of ANNs 15
2.2.1 Determining the structure of ANNs 15
2.2.2 Weight initialization 18
2.2.3 Optimizer 18
2.3 Pei et al. 20
2.4 Robotic arm and target equations 24
2.4.1 Simplified PUMA robotic arm 25
2.4.2 Target equations 25

3 Research Questions **28**
3.1 Structural issues 29
3.1.1 Research question 1.1 - Structural transferability 29

3.1.2 Research question 1.2 - Extensibility to more hidden layers 30
3.2 Initialization 31
3.2.1 Research question 2.1 - Domain transferability 31
3.2.2 Research question 2.2 - Concatenating prototypes 32
3.2.3 Research question 2.3 - Performance evaluation 33
3.3 Other Issues 33
3.3.1 Research Question 3.1 - Robustness to noise 33

4 Experiments 35
4.1 Data generation 35
4.2 Experimental design 37
4.3 Prototypes used 38
4.3.1 Approximating constant - Prototype A 40
4.3.2 Approximating arithmetic and polynomial operations - Prototype B 40
4.3.3 Approximating cosine curves - Prototype C 41
4.4 Initial architecture and parameter values for I^3 44
4.4.1 Approximating Equation 2.3: $\hat{x} = 0.5\cos(2\pi\hat{\theta}_1) + 0.5$ . . . 45
4.4.2 Approximating Equation 2.4: $\hat{y} = 0.5\sin(2\pi\hat{\theta}_1) + 0.5$ . . . 48
4.4.3 Approximating Equation 2.10 50
4.5 Other architectures used 55
4.5.1 Architecture to approximate Equation 2.13 $\hat{y} = 0.5\sin(2\pi\hat{\theta}_1) * \cos(2\pi(\hat{\theta}_3 + 1/8)) + 0.5$ 55
4.5.2 Architecture to approximate Equation 2.14 $\hat{x} = -0.25\cos(2\pi\hat{\theta}_1) * [\sin(2\pi(\hat{\theta}_2 + \hat{\theta}_3)) - cos(2\pi\hat{\theta}_2)] + 0.5$ 59
4.6 Tools and software used 59

5 Results 61

6 Discussion 66
6.1 Structural issues 66
6.1.1 Structural transferability 66
6.1.2 Extensibility to more hidden layers 67
6.2 Initialization issues 67
6.2.1 Domain transferability 67
6.2.2 Concatenating prototypes 68
6.2.3 Performance evaluation 71
6.3 Effect of noise 73
6.3.1 Robustness to noise 73

7 Conclusions 74

8 Future Work **76**
8.1 Theoretical advancements 76
8.2 Practical advancements 77

Bibliography **79**

A Appendix **83**
A.1 Initial parameters for other equations using I^3 83

List of Figures

1.1 Generic artificial neural network where biases are not included. p_1, p_2 are inputs, H corresponds to activation function of hidden layer, and O corresponds to activation function of output layer. . 2

2.1 Taxonomy of informed machine learning as directly adopted from (Von Rueden et al., 2021), where the current work is highlighted in green, in the backdrop of existing main paths shown in gray. 9
2.2 Frequency of term artificial intelligence (AI) and machine learning (ML) in the English language corpus (Source: Google books ngram viewer). 13
2.3 Frequency of term explainable artificial intelligence (XAI) and interpretable machine learning (IML) in the English language corpus (Source: Google books ngram viewer). 13
2.4 Taxonomy of IML in ANNs based on three dimensions adopted from (Zhang et al., 2021), where the current work is highlighted in green. 14
2.5 Sigmoid function. 22
2.6 ANN architecture used to approximate a constant term. Nodes labeled S and L have sigmoidal and linear activation functions, respectively. 22
2.7 Approximating constant (1) as linear sum of two sigmoidal units. 23
2.8 3-D orientation of simplified PUMA robotic arm. 26

4.1 Prototype C used to approximate cosine term. 42
4.2 $\hat{x}$ to be approximated given $\hat{\theta}_1$ (a) Target function to be approximated (b) Prototype function obtained using I^3 to approximate target function. 45
4.3 ANN architecture with one hidden layer and four hidden nodes. S, L nodes correspond to sigmoidal, linear activation functions respectively. 47
4.4 Prototype 2 used to approximate sine term. 48

4.5 $\hat{y}$ to be approximated given $\hat{\theta}_1$ (a) Target function to be approximated (b) Prototype function obtained using I^3 to approximate target function. 48
4.6 ANN architecture to approximate term 1. S, L nodes correspond to sigmoidal, linear activation functions respectively. 50
4.7 Black is for target function, while red is the approximated function (without training). 51
4.8 Non condensed ANN architecture to approximate term 2. S, L nodes correspond to sigmoidal, linear activation functions respectively. 54
4.9 Condensed ANN architecture to approximate term 2. S, L nodes correspond to sigmoidal, linear activation functions respectively. . . 54
4.10 ANN architecture to approximate term 3. Nodes labeled S and L have sigmoidal and linear activation functions, respectively and p can be any input/value 55
4.11 ANN architecture before condensing hidden layer - for $\hat{\theta}_2, \hat{\theta}_3$ as inputs and $\hat{x}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively. 56
4.12 ANN architecture after condensing hidden layer - for $\hat{\theta}_2, \hat{\theta}_3$ as inputs and $\hat{x}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively. 57
4.13 ANN architecture after condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_3$ as inputs and $\hat{y}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively. 58
4.14 ANN architecture after condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3$ as inputs and $\hat{x}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively. 60

5.1 Scatter plot of prototypical example of learning a function (a) Excellent learning (b) Moderate learning (c) NW Learning poorly . . 64

6.1 Initial errors reported for $\hat{\theta}_1$ as input and $\hat{x}$ as output. 68
6.2 Initial errors reported for $\hat{\theta}_1$,$\hat{\theta}_3$ as input and $\hat{y}$ as output. 69
6.3 History of error Vs epochs for Equation 2.13. The spike in the black rectangular box indicates the large updates in ANN weights and biases during training. 70
6.4 History of error Vs epochs for Equation 2.14. The spike in the black rectangular box indicates the large updates in ANN weights and biases during training. 70
6.5 Errors reported while training across all repetitions (a) $\hat{y}$ given $\hat{\theta}_1$ (b) $\hat{x}$ given $\hat{\theta}_2$. 71
6.6 History of errors when NW stuck at local optima while training $\hat{y}$ given $\hat{\theta}_1$. 72

A.1 ANN architecture before condensing hidden layer. S, L nodes correspond to sigmoidal, linear activation functions respectively. . 85
A.2 ANN architecture after condensing hidden layer. S, L nodes correspond to sigmoidal, linear activation functions respectively. . . . 85
A.3 $\hat{x}$ to be approximated given $\hat{\theta}_3$ (a) Target function to be approximated (b) Prototype function obtained using I^3 to approximate target function. 87
A.4 ANN architecture before condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_2$ as inputs. S, L nodes correspond to sigmoidal, linear activation functions respectively. 91
A.5 ANN architecture after condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_2$ as inputs. S, L nodes correspond to sigmoidal, linear activation functions respectively. 92
A.6 ANN architecture before condensing hidden layer - for $\hat{\theta}_2, \hat{\theta}_3$ as inputs and $\hat{z}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively. 96
A.7 ANN architecture after condensing hidden layer - for $\hat{\theta}_2, \hat{\theta}_3$ as inputs and $\hat{z}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively. 97
A.8 ANN architecture before condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_3$ as inputs and $\hat{x}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively. 101
A.9 ANN architecture after condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_3$ as inputs and $\hat{x}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively. 102
A.10 ANN architecture before condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_3$ as inputs and $\hat{y}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively. 104
A.11 ANN architecture after condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_3$ as inputs and $\hat{y}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively. 105

List of Tables

2.1 Derived weights and biases in approximating the first two polynomials (Pei et al., 2005) and the basic arithmetic operations of addition and multiplication (Pei et al., 2013). $w^{[0]}$ and $w^{[1]}$ are the arbitary values used while designing. 21

2.2 Equations for different input-output combinations derived using forward kinematics of robotic arm. In the kinematics equations, the joints has only one cycle $(0 - 2\pi)$. *Note:* Some of the input-output combinations are excluded from the table as they either have constant value or have the same as other existing input-output combination. 27

5.1 Comparison of models with two different initializations. For Noise level, N, L, M, and H correspond to no, low, medium, and high noise, respectively. Best Error is the smallest error reported in any repetition from either approach based on the architecture suggested by the I^3. M-W p-value is the value of the Mann-Whitney U test. 62

5.2 Comparison of models with two different initializations (Kanneganti et al., 2021). For Noise level, N, L, M, and H correspond to no, low, medium, and high noise, respectively. M-W $p - value_{NW}$ is the p-value of the Mann-Whitney U test for NW and I^3. 65

Chapter 1

Introduction

This chapter introduces artificial neural networks (ANNs), informed machine learning, interactive machine learning and interpretable machine learning (IML). Outlines the Pei et al. (2001 to 2022) on which this dissertation is built. This chapter also introduces the motivations and contributions of this dissertation, the domain (robot kinematics) on which Pei et al. was extended.

1.1 Introduction to interpretable machine learning

Machine learning (ML) is a subset of artificial intelligence (AI) that allows software applications to understand and discover patterns from data or experiences and makes accurate predictions without being explicitly programmed to do so. *Artificial neural networks* (ANNs) are computational models implemented in software or custom-made hardware devices that attempt to capture the behavioral and adaptive features of biological nervous systems. ANNs are subset of ML

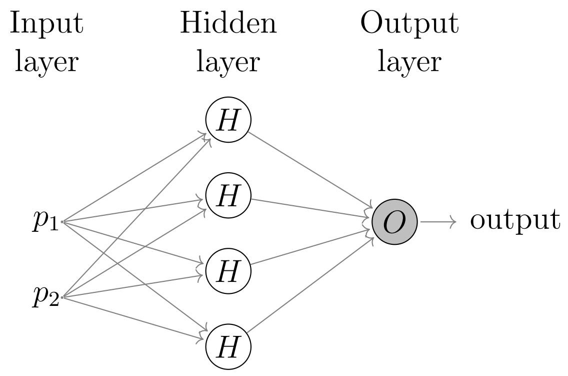


Figure 1.1: Generic artificial neural network where biases are not included. p_1, p_2 are inputs, H corresponds to activation function of hidden layer, and O corresponds to activation function of output layer.

algorithms. An ANN is composed of several interconnected units, nodes, or neurons as in Fig. 1.1. Some of these units receive information directly from the environment (input units), some have a direct effect on the environment (output units), and others communicate only with units within the network (internal, or hidden, units) (Floreano and Mattiussi, 2008).

ANNs are universal function approximators (Cybenko, 1989; Hornik, 1991), i.e., the multilayer ANNs with as few as one hidden layer can represent a wide variety of interesting functions when given appropriate weights. By having sufficiently many nodes in its hidden layer(s), any continuous nonlinear function can be learned by sigmoidal ANNs to any desired degree of accuracy. Unfortunately, standard ANNs are opaque because there is no inherent meaning to the hidden nodes. This makes it difficult to understand the model and restricts its use in making important decisions. To bridge this knowledge gap for modeling kinematics, we use *interpretable machine learning*, i.e., ML in which the response of the model given the input is inherently understandable (Rudin, 2019). We begin by recognizing existing research (Pei et al.) in nonlinear function approximation. This long-term multidisciplinary research effort has made strides toward providing end users of ANNs with interpretable modeling tools for engineered systems (Pei, 2001; Pei et al., 2005, 2013; Pei and Mai, 2008; Pei et al., 2008, 2021).

In Pei et al., ANNs are engineered as transparent, linear-in-the-weight parameterization tools by using domain knowledge/constraints and visual features to determine appropriate basis functions. With few hidden nodes to activate these basis functions, the modeling capability of sigmoidal ANNs is systematically revealed without relying on training. The key is the developed ANN initialization theory and practical procedures that can be interpreted via domain knowledge. A rich repository of direct (non-iterative) techniques and reusable ANN prototypes can be aggregated as the basis functions needed for specific problems, leading to interpretable ANNs as well as improved training performance and generalization as validated by simulated and real-world data (Pei, 2001; Pei et al., 2005; Pei and Mai, 2008; Pei et al., 2008, 2013, 2021).

In this work we have reintroduced, highlighted, and established the larger picture to Pei et al. in the context of being informative, interactive, and interpretable. As we develop this I^3 framework, it is a general framework for bringing together the ideas of being informed, interactive, and interpretable. Previous work by Pei et al. can be interpreted as a candidate in this I^3 framework as it satisfies the above three components. Throughout the document, for brevity we will refer to Pei et al. as I^3. But Pei et al. is one instantiation of I^3 framework and this framework itself could be instantiated in other ways.

1.1.1 Informative

Additional integration of prior knowledge into the training process is called *informed ML* (Von Rueden et al., 2021). I^3 is informed as I^3 involves integrating prior knowledge:

1. The basis functions suggested by domain knowledge.
2. To examine dominant features from visualization of data.

More about this is explained with an example in Section 2.1.1 and Section 4.4.3.

1.1.2 Interactive

Interactive ML enables ML models to be interactively steered by humans and is more advantageous for the tasks where human knowledge is needed in the analysis process (Jiang et al., 2019). I^3 is interactive as it involves:

1. Choosing appropriate prototype and variant.
2. Adjust certain weights and biases in selected prototype.

More about this is explained in Section 2.1.2, and with an example in Section 4.4.3.

1.1.3 Interpretable

IML is an young field and there is no unified definition for interpretability (Lipton, 2018; Molnar et al., 2020; Murdoch et al., 2019). In this work we refer to interpretability as having meaning to each of its components. Interpretability is the result of being informative and interactive. The interpretability lies in:

1. To have meaning to each of its components.
2. The similarity between the output of the initial ANN and the target function.

This approach is used to model forward kinematics of robot arms and thus the title “Informed, interactive, and interpretable machine learning for forward kinematics of robot arms”.

1.2 Introduction to robot kinematics and robotic arm used

Experiments were conducted using multiple nonlinear datasets based on a simplified model of a PUMA (Programmable Universal Manipulation Arm or Programmable Universal Machine for Assembly) robot arm. PUMA is a serial robot and it consists of sequence of links and joints.

Kinematics describes the relationship between a robot’s joint coordinates and its spatial layout. It is a fundamental and classical topic in robotics. Kinematics calculations are used for tasks such as positioning a gripper at a position in space, designing a mechanism to move a tool between two points, and predicting if robot’s motion would collide with obstacles. Kinematics is concerned with only the instantaneous values of a robot’s coordinates and is not concerned about the movements under forces and torques.

In this work, we are only concerned with *forward kinematics*—using the kinematic equations to calculate the positions of the robot’s links from specified values for the joints (Hauser, 2020). In our experiments, we design models for predicting the end position in X, Y, and Z three-space given normalized waist ($\hat{\theta}_1$), shoulder ($\hat{\theta}_2$), and elbow ($\hat{\theta}_3$) angles. This research is only carried out using carefully simulated datasets. More about this robotic arm is discussed in Section 2.4.

1.3 Intended contributions

This work is the first to use this I^3 approach for robotics. **The intended contributions of this work are:**

1. **This work sees the power of informed, interactive, and interpretable ML, and identifies Pei et al. being informed, interactive, and interpretable. This work reintroduces, validates, facilities Pei et al. to be a full-fledged informed, interactive, and interpretable approach. Bringing together the ideas of informed, interactive, and interpretable ML makes this work more accessible for broader audience.**

2. **Developing the terminology of I^3 that tries to mean in accordance with the ideas that are put together. Developing the terminology helps to make Pei et al. connected to other areas and more accessible for broader audience.**

3. **Even though Pei et al. has done multi-disciplinary research, applications are mainly limited to engineering mechanics. The work on epidemic curves (Hougen et al., 2020) and (Pei et al., 2021) are in the middle of the transition from engineering mechanics domain to computer science domain. This present work is from computer science ML view point. Tremendous effort is made to select and refine the terminologies. This helps to make this ap-**

proach understandable by different audience and strengthen the approach by contributing ideas from other perspectives.

4. **To demonstrate the feasibility of transferring the I^3 to a different application (robot kinematics). Robot kinematics is chosen as a flexible example for how to transfer this to a new domain.**

5. **To demonstrate the feasibility and determine the challenges of using I^3 to develop and train ANNs with more inputs. Scaling up inputs allows us to model muti-dimensional input spaces.**

6. **To demonstrate the feasibility and determine the challenges of using I^3 to develop and train ANNs with more hidden layers when needed. More hidden layers allows for the modeling of more complex functions.**

7. **Earlier I^3 has used Levenberg-Marquardt as an optimizer. In this work we have used I^3 using Adam as an optimizer. Using Adam facilitates the way for training deeper ANNs. More about this is discussed in Section 2.2.3.**

The remainder of this dissertation is organized as follows: The overview of I^3 and other related work is explained in Chapter 2. Research questions and hypothesis are discussed in Chapter 3. The experimental setup, the data used, the architectures and initial values of target equations are explained in Chapter 4. The results and discussion are detailed in Chapter 5 and Chapter 6, respectively. Conclusions of this work are mentioned in Chapter 7 and future work is mentioned in Chapter 8.

Chapter 2

Related Work

This chapter explains background of informed, interactive, and interpretable ML and also explains some of the fundamentals of ANNs, an overview of the existing work and the foundation on which this research is built. This section details the robotic arm used and outlines the target equations that will be modeled later.

2.1 Background of informed, interactive, and interpretable ML

This section discusses the background of the three ingredients proposed in I^3 - informed, interactive, and interpretable ML.

2.1.1 Informed ML

Despite its success, ML algorithms have limits when there is insufficient amount of training data. A potential solution is the additional integration of prior knowledge into the training process which leads to the notion of informed machine

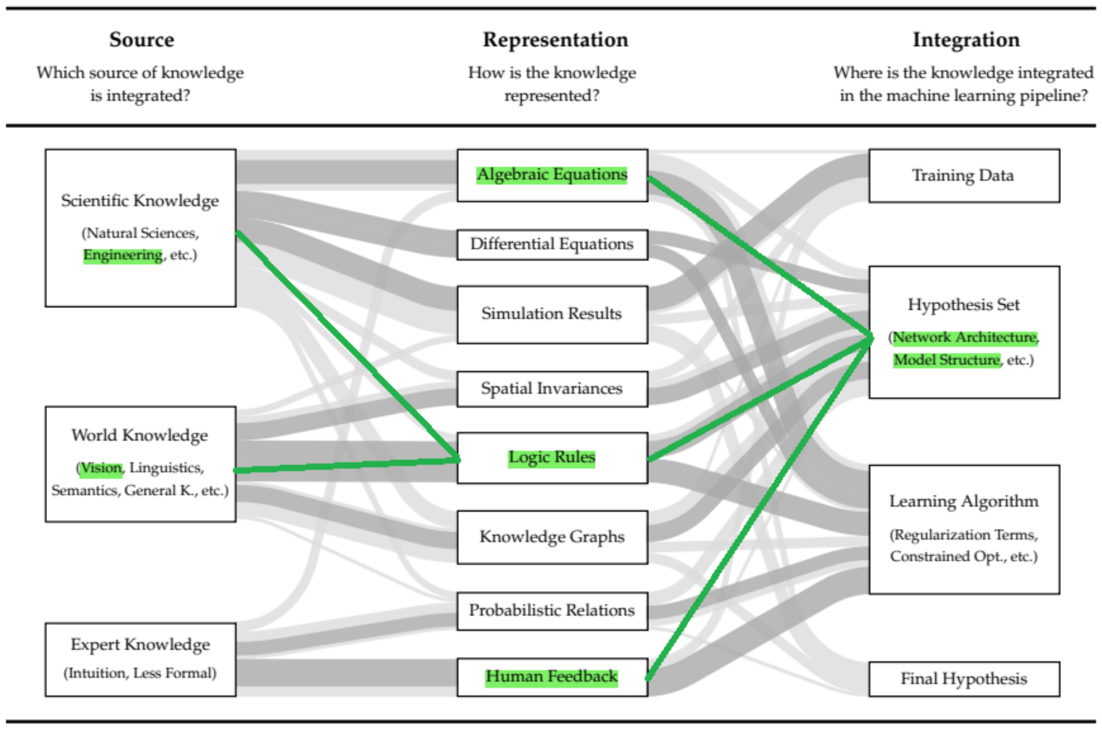

Source
Which source of knowledge is integrated?
Representation
How is the knowledge represented?
Integration
Where is the knowledge integrated in the machine learning pipeline?
Scientific Knowledge
(Natural Sciences, Engineering, etc.)
World Knowledge
(Vision, Linguistics, Semantics, General K., etc.)
Expert Knowledge
(Intuition, Less Formal)
Algebraic Equations
Differential Equations
Simulation Results
Spatial Invariances
Logic Rules
Knowledge Graphs
Probabilistic Relations
Human Feedback
Training Data
Hypothesis Set
(Network Architecture, Model Structure, etc.)
Learning Algorithm
(Regularization Terms, Constrained Opt., etc.)
Final Hypothesis

Figure 2.1: Taxonomy of informed machine learning as directly adopted from (Von Rueden et al., 2021), where the current work is highlighted in green, in the backdrop of existing main paths shown in gray.

learning (Von Rueden et al., 2021). Figure 2.1 shows the taxonomy of informed machine learning. The taxonomy consists of the three dimensions knowledge source, knowledge representation, and knowledge integration. Source informs the source of knowledge that is integrated. Representation informs how knowledge is represented. Integration informs where in the learning pipeline knowledge is integrated. Informed machine learning describes learning from a hybrid information source that consists of data and prior knowledge. The prior knowledge comes from an independent source, is given by formal representations, and is explicitly integrated into the machine learning pipeline.

Each dimension contains a set of elements that represent the spectrum of different approaches found in the literature as illustrated in Fig. 2.1. There are three different sources of knowledge. They are scientific knowledge, world knowledge, and expert knowledge. Scientific Knowledge is from the subjects of science, technology, engineering, and mathematics and this knowledge is typically formalized and validated explicitly through scientific experiments. World knowledge refers to facts from everyday life that are known to almost everyone and can thus also be called general knowledge. It can be more or less formal. Generally, it can be intuitive and validated implicitly by humans reasoning in the world surrounding them. Expert knowledge is the knowledge that is held by a particular group of experts.

The I^3 uses two sources, scientific knowledge obtained through applied mathematics and engineering (I^3 is first developed using applied mathematics for engineering mechanics) and world knowledge through vision (in I^3 dominant features in the data are visualized). More about this is discussed in Section 2.3.

The second dimension is knowledge representation and knowledge can be represented in one or more of the following representations: algebraic equations, differential equations, simulation results, spatial invariances, logic rules, knowledge graphs, probabilistic relations, and human feedback.

In I^3 knowledge is represented in algebraic equations (knowledge represented in polynomial functions, arithmetic equations, etc.), logic rules (selecting prototype and variant from visually observable features), and through human feedback (choosing appropriate prototype and variant, and adjusting weights and biases of ANNs in selected prototype).

There are four different ways of integrating knowledge in ML pipeline. They are in training data (incorporating knowledge in the training data), hypothesis set (knowledge integrated to define ANN architecture and hyper-parameters), learning algorithm (knowledge integrated to modify loss function), and final hypothesis (the output of model can be validated against existing knowledge). I^3 helps in determining the structure of ANN, so knowledge is integrated in hypothesis set.

So, the I^3 is an informed machine learning approach, where the scientific knowledge through engineering and the world knowledge through vision is represented in the form of algebraic equations, logic rules and through human feedback. And the represented knowledge helps to narrow down the hypothesis for network architecture.

2.1.2 Interactive ML

The typical interactive ML workflow includes two steps in each iteration. First, the intermediate results of the model as well as the data is visually presented to the user, the user explores the visualization results, draw understandings and insights about the data and the model, then input feedback to the model. Second, the model is incrementally updated by integrating the human input (Jiang et al., 2019). Hence, interactive ML enables ML models to be interactively steered by humans and is more advantageous for the tasks where human knowledge is needed in the analysis process (Jiang et al., 2019). Some of those approaches are visual cluster analysis, interactive dimensionality reduction, etc.

In order to be interactive I^3 brings the world knowledge from informed aspect. In I^3 visualization of data, choosing prototype and variant, and adjusting weights and biases of ANNs in selected prototype are done prior to any modeling and helps in determining appropriate ANN architecture. More about prototype and I^3 is explained in Section 2.3. This approach is slightly different from the definition explained above, where visualization (interactive component) is done on the intermediate results of the model when compared to I^3 where visualization is done before modeling; For instance, in cluster analysis and in dimensionality reduction visualization is performed after modeling (clustering or reducing the dimension of data). I^3 also integrates the human feedback by choosing appropriate prototype and variant, and adjusting weights and biases of ANNs in selected prototype.

2.1.3 Interpretable ML

Figure 2.2 shows the level of interest in AI and ML. The level of interest in AI picked up around 1960 and reached to its peak around 1990, then the interest in AI decreased till 2014 and then kept increasing again. The level of interest in ML picked up after 1980 and exploded after 2012. It could be this explosion that is recreating interest in AI. Figure 2.3 shows the level of interest in Explainable AI (XAI) and IML. The level of interest in XAI picked up around 2002 and reached to its peak around 2009, then the interest in XAI decreased till 2014 and then

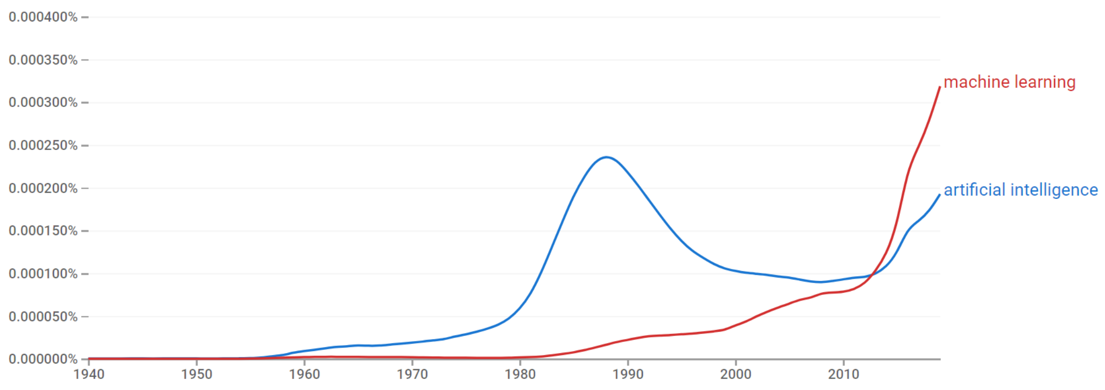


Figure 2.2: Frequency of term artificial intelligence (AI) and machine learning (ML) in the English language corpus (Source: Google books ngram viewer).

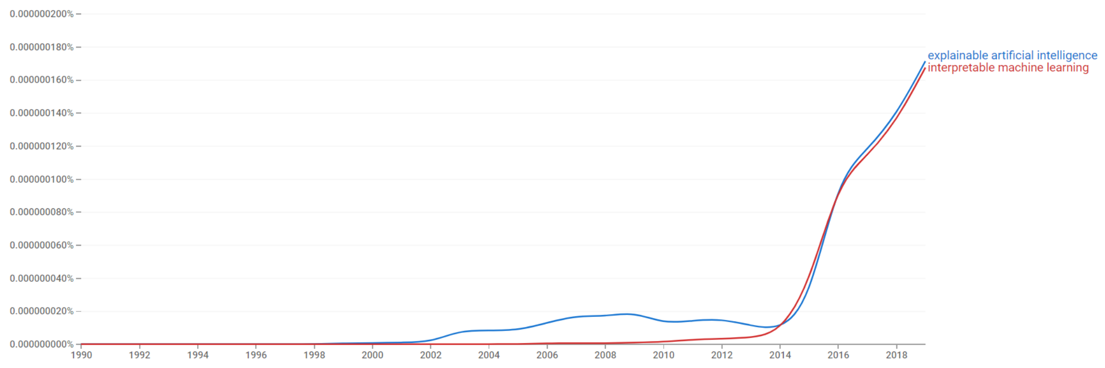
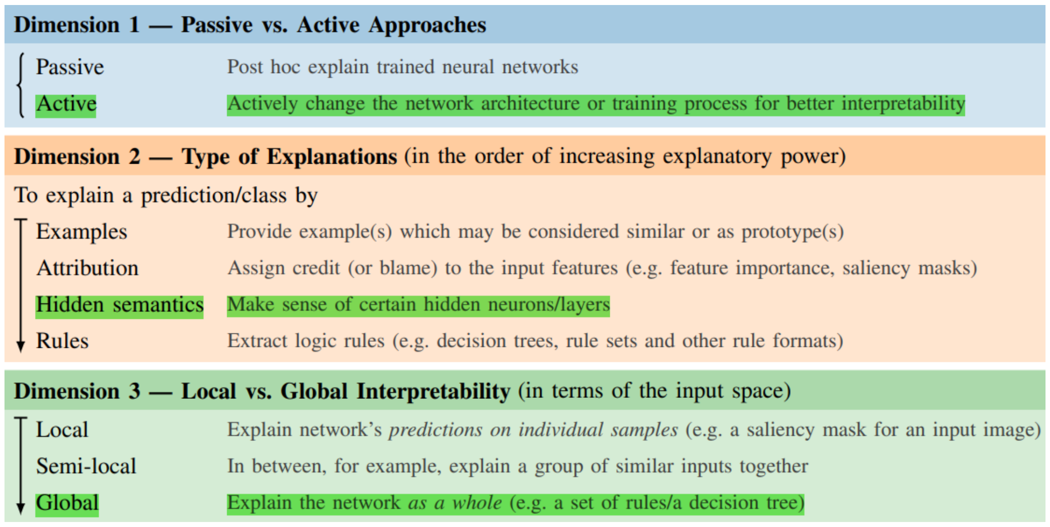


Figure 2.3: Frequency of term explainable artificial intelligence (XAI) and interpretable machine learning (IML) in the English language corpus (Source: Google books ngram viewer).

kept increasing again. The level of interest in IML picked up after 2012. It is this increase in interest in IML that drives interest in XAI.

By observing Fig. 2.2 and Fig. 2.3 we can infer that most of the earlier ML research has focused on improving evaluation metrics to the exclusion of other qualities like interpretability. Recently, adoption of IML models or systems has been expanding (Carvalho et al., 2019). IML is an young field and there is no unified definition for interpretability (Lipton, 2018; Molnar et al., 2020; Murdoch et al., 2019). One definition of interpretability is the ability to provide explanations in understandable terms to humans (Doshi-Velez and Kim, 2017). One



Dimension 1 — Passive vs. Active Approaches	
Passive	Post hoc explain trained neural networks
Active	Actively change the network architecture or training process for better interpretability
Dimension 2 — Type of Explanations (in the order of increasing explanatory power)	
To explain a prediction/class by	
Examples	Provide example(s) which may be considered similar or as prototype(s)
Attribution	Assign credit (or blame) to the input features (e.g. feature importance, saliency masks)
Hidden semantics	Make sense of certain hidden neurons/layers
Rules	Extract logic rules (e.g. decision trees, rule sets and other rule formats)
Dimension 3 — Local vs. Global Interpretability (in terms of the input space)	
Local	Explain network's *predictions on individual samples* (e.g. a saliency mask for an input image)
Semi-local	In between, for example, explain a group of similar inputs together
Global	Explain the network *as a whole* (e.g. a set of rules/a decision tree)

Figure 2.4: Taxonomy of IML in ANNs based on three dimensions adopted from (Zhang et al., 2021), where the current work is highlighted in green.

approach to IML is to have meaning to each of its components; In this work we refer to this meaning for interpretability. I^3 is interpretable because of informed and interactive nature of this approach.

With increase in success of deep neural networks, there is also increase in concern about their black-box nature (Zhang et al., 2021). Taxonomy of IML in ANNs based on three dimensions is discussed in (Zhang et al., 2021) and shown in Fig. 2.4. One of the challenges of IML models is evaluation of interpretability. Some of the challenges of IML are discussed in (Rudin et al., 2022; Vollert et al., 2021).

First dimension to categorize IML is by *active* and *passive* approaches. Active approaches change the network architecture or training process for better interpretability. Passive approaches have *post hoc* explanations for trained ANNs. Post hoc explanations provide interpretability after a model has been developed. Second dimension to categorize IML is by type of explanations the models provide. Third dimension is *local* vs. *global* interpretability. Local interpretability refers to explaining ANN's predictions on individual samples, global interpretability refers to explaining network as a whole, and semi-local refers to explaining group of similar inputs together.

I^3 is an active approach, explains hidden semantics, and has global interpretability. I^3 is active as ANN architecture is designed based on domain knowledge and by visualizing data, explains hidden semantics as hidden nodes and layers have meaning in the ANN, has global interpretability as we are explaining the ANN as a whole rather than explaining for individual or group of similar inputs.

2.2 Fundamentals of ANNs

This section discusses about determining structure of ANNs, weight initialization approaches, and the optimizers.

2.2.1 Determining the structure of ANNs

There are multiple approaches in determining the structure of ANNs. One of the ways is to try to use heuristics or rules of thumb. Another way is to iteratively (increasing/decreasing number of hidden layers/nodes by trail and error) try to determine the correct structure of the ANN.

Heuristic approaches

Below are three approaches for determining the structure of ANN using heuristic approach:

1. There are few approaches that suggest the structure of neural networks. Chapter 5.9 of (Kruse et al., 2011) suggests to use (number of inputs +

number of outputs)/2 hidden units in case of single hidden layer. Using this approach, for equations with one input and one output just requires one hidden unit. Similarly, for equations with two inputs and one output just requires one or two hidden units. But these are very less number of hidden units and so this guidance is not applicable to approximate equations of robotic arms.

2. Chapter 7.1 of Mehlig (2021) discusses splitting target function as a combination of basis functions. This approach recommends using 2 hidden nodes for each basis function. However, it does not discuss the approach to obtain the number of basis functions. However, this approach suggests a large number of basis functions which results in larger number of hidden nodes for sine or cosine functions, which is related to the equations of robotic arms.

3. Approach in (Jones, 1997) discusses about approximating continuous functions using a linear combination of sigmoidal functions. The author mentioned that any continuous function can be approximated by a Finite Fourier cosine series and a function can be approximated using the difference of two non-decreasing functions. Thus a linear combination of sigmoidal functions/ridge functions can be helpful in approximating a continuous function. For a given function, we can have a maximum error of $1/i$, where i is the even number of neurons; So, by taking i sufficiently large the construction may be made sufficiently accurate. This approach mentions the relationship between the upper bound of error and the number of neurons. This paper did not explicitly mention the architecture required for a given target function; However, in this approach author has qualitatively explained the

number of neurons w.r.t performance.

Iterative approaches

Iterative approaches has different ways of iteratively trying to determine the structure of ANN by increasing/decreasing number of hidden layers/nodes by trail and error. Some such mechanisms are listed below:

1. Start with a small structure and keep growing the structure and stop at an error we are hoping at.
2. Start with a large network and reduce the size of network.

Other approaches

1. Bayesian regularization is a special case of iterative approach. This approach starts with a large structure. After training, some of the weights goes to zero. The number of non zero weights (effective weights) are less than total available weights. This approach helps to reduce the size of ANN to the effective number of parameters. This is a principle pruning approach. This helps to get a good estimate of number of weights required to approximate the target function (Demuth et al., 2014).

Pei et al.

The I^3 brings more knowledge (domain knowledge and visually dominant features) to determine the structure of ANNs while approximating the target function.

2.2.2 Weight initialization

ANNs can be viewed as a function with learnable parameters (weights and biases). The performance of ANNs, convergence to optimal solution depends on how the parameters are initialized before the start of training. Thus, a better way of initialization is required for modeling ANNs. Two popular initialization techniques are discussed below: Nguyen-Widrow (Nguyen and Widrow, 1990) and Xavier (Glorot and Bengio, 2010) initializations.

In *Nguyen-Widrow* (NW) initialization, weights are assigned in such a manner that the region of interest is divided into small intervals. The training process speeds up by setting the initial weights of the hidden layer, so that each hidden node is assigned its own interval at the start of training. The network is trained with each hidden node still having the freedom to adjust its interval size and location during the training. Thus, the NW approach generates initial weights and bias values for a layer such that the active regions of the layer's neurons are distributed approximately evenly over the input space (Nguyen and Widrow, 1990). In case of sigmoidal ANNs, the weights converge quicker when the input to the activation function falls in the region having more slope. *Xavier initialization* (Glorot and Bengio, 2010) sets the initial values of parameters in such a way that aids for quicker convergence and better performance in ANNs. This approach is quickly adopted and used as a state-of-the-art initialization technique.

2.2.3 Optimizer

An *optimizer* is a method or algorithm that changes the attributes of a neural network, such as weights, biases, and learning rate. This helps in reducing overall loss and decrease the error or improve the accuracy. There are many optimizers but Adaptive Moment Estimation (Adam) (Kingma and Ba, 2017) is generally used for deep learning. Levenberg–Marquardt (Moré, 1978) is preferred for training shallow ANNs especially when error is mean squared error (MSE).

Adam is an extension of stochastic gradient descent (SGD) and is used to update network parameters while training. Unlike using a single learning rate for SGD, Adam computes adaptive learning rates for each parameter, i.e., Adam optimizer updates the learning rate for each network weight individually. Adam has both the benefits of Adagrad and RMSProp (Root Mean Square) algorithms, both of these algorithms are also an extension of SGD algorithms (Kingma and Ba, 2017). This has been used as a default optimizer in most of deep learning models. Adam has proven to be efficient algorithm interms of runtime and memory usage.

Levenberg–Marquardt (LM) commonly known as Damped Least Squares method and is used to solve non-linear least squares problems. Least squares problems arise in the context of fitting a parameterized mathematical model to a set of data points by minimizing an objective expressed as the sum of the squares of the errors between the model function and a set of data points. The LM uses two different minimization algorithms: the gradient descent and the Gauss-newton method. In the gradient descent method, the sum of the squared errors is reduced by updating the parameters in the steepest-descent direction. In the Gauss-Newton method, the sum of the squared errors is reduced by assuming the least squares function is locally quadratic in the parameters, and finding the minimum of this quadratic. The LM acts more like a gradient-descent method when the parameters are far from their optimal value, and acts more like the Gauss-Newton method when the parameters are close to their optimal value (Gavin, 2019).

2.3 Pei et al.

Traditional ML algorithms are opaque. IML is transparent as it injects domain knowledge into ML design. This results in a system that is understandable and trustworthy. Early IML research for pattern classification using ANNs states, “the first layer partitions the input space into a number of cells. The sole function of additional layers is then to group these cells into decision regions” (Makhoul et al., 1989). This fundamental idea can also be found in deep learning (LeCun et al., 2015) and explainable AI (XAI) (Gunning et al., 2019). This fundamental idea is the basis of designing transparent sigmoidal neural networks for function approximation.

In the work reviewed here (Pei, 2001; Pei et al., 2005, 2013; Pei and Mai, 2008; Pei et al., 2008, 2021), static nonlinear functions are approximated using feedforward ANNs, using sigmoidal activation functions with a one or two hidden layer(s), and taking an input vector to produce a scalar output. I^3’s fundamental approach is applied mathematics and has the potential of being useful beyond engineering mechanics. To design interpretable neural networks, I^3 uses domain knowledge including visually observable features to determine (a) structure of the network (including the number of hidden nodes) (b) initial values of the parameters (weights and biases in the network) as the first approximation of the function to be learned, and then trained on the data for local optimization. These initial parameters are obtained based on theory and not from training. Determining structure and initial values of ANNs is a longstanding question (Sontag, 1992) that remains open in general.

Structure, parameter values to carry out basic computing operations and polynomial functions with certain approximation accuracy have been derived in earlier work, including the number of hidden nodes, weights, and biases for approximating four basic arithmetic operations (Pei et al., 2013) and the number of hidden nodes, weights, and biases for approximating polynomial terms p^0, p^1, p^2, and p^3 (Pei et al., 2005). Table 2.1 has derived values of the weights and biases for some of the functions that are used in this dissertation. In Table 2.2, $p_1 \times p_2$ has larger weights to ensure low error. All the functions can be approximated with just one hidden layer.

Table 2.1: Derived weights and biases in approximating the first two polynomials (Pei et al., 2005) and the basic arithmetic operations of addition and multiplication (Pei et al., 2013). $w^{[0]}$ and $w^{[1]}$ are the arbitary values used while designing.

Target Function	Weights in Input Layer, W_1	Biases in Input Layer, b	Weights in Output Layer, W_2
p^0	$\begin{bmatrix} w^{[0]} \\ -w^{[0]} \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \end{bmatrix}$
p^1	$\begin{bmatrix} w^{[1]} \\ -w^{[1]} \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} +\frac{2}{w^{[1]}} & -\frac{2}{w^{[1]}} \end{bmatrix}$
$p_1 + p_2$	$\begin{bmatrix} 0.1 & 0 \\ -0.1 & 0 \\ 0 & 0.1 \\ 0 & -0.1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 20 \\ -20 \\ 20 \\ -20 \end{bmatrix}^T$
$p_1 \times p_2$	$\begin{bmatrix} 0.1 & 0.1 \\ -0.1 & -0.1 \\ 1 & 1 \\ -1 & -1 \\ 0.1 & -0.1 \\ -0.1 & 0.1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix}$	$\begin{bmatrix} -10 \\ -10 \\ 0 \\ 0 \\ -10 \\ -10 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 5.5076165 \times 10^5 \\ 5.5076165 \times 10^5 \\ -50.006810 \\ -50.006810 \\ -5.5076165 \times 10^5 \\ -5.5076165 \times 10^5 \\ 50.006810 \\ 50.006810 \end{bmatrix}^T$

The author is sharing their understanding as follows, below is an example to demonstrate approximating a constant term ($p^0 = 1$) using two logistic sigmoidal units (S) where

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

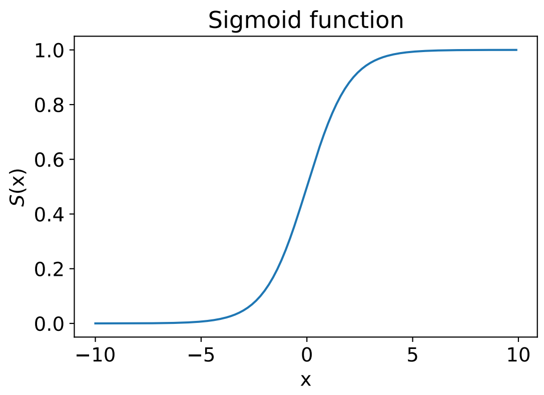


Figure 2.5: Sigmoid function.

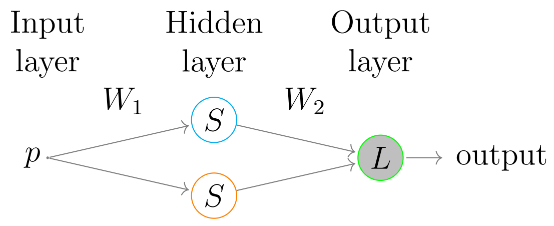


Figure 2.6: ANN architecture used to approximate a constant term. Nodes labeled S and L have sigmoidal and linear activation functions, respectively.

where $x = W^T \times P + b$, where W^T is the transpose of the weight matrix, P is the input to the sigmoidal unit and b is the bias. Output of S can be observed in Fig. 2.5.

In Fig. 2.6, W_1 is the array of weights from the input layer to the hidden layer and W_2 is the weight array from the hidden layer to the output layer. Likewise, b_1 is the bias array of the hidden nodes and b_2 is the bias of the output node.

From Table 2.1, W_1 could be $[1, -1]$, and W_2 has the value of constant we would like to approximate, so W_2 becomes $[1, 1]$, b_1 takes the values of $[0, 0]$, and b_2 equals $[0]$. Considering p as the input, the output takes the value of $1 \times (S(p \times 1 + 0)) + 1 \times (S(p \times -1 + 0)) + 0$. Solving this gives an output of 1. Figure 2.7 also demonstrates approximating a constant, we can observe the output of both the hidden nodes and the linear sum of these two hidden nodes

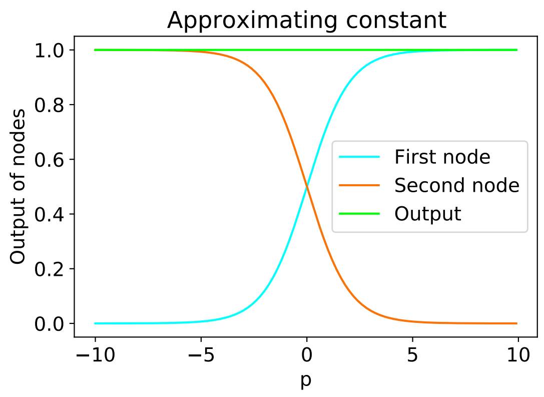


Figure 2.7: Approximating constant (1) as linear sum of two sigmoidal units.

can be observed at output node and it equals to 1.

This approach can be used to approximate any constant value. Thus, I^3 provides a meaningful interpretation of nodes used in the ANNs, thus comes under IML. In this work, I^3 initialization refers to the parameter determination using I^3 approach explained in this section.

Domain knowledge provides useful information. First, I^3 tries to answer these questions: (a) What are the useful relations between input and output? One way of understanding these relations is by visually observing the features of the data. (b) What sigmoidal neural network prototypes can be used for each useful feature? Prototypes and their variants are predetermined ANNs that are constructed in advance from a forward formulation based on either algebraic or geometric capabilities of linear sums of sigmoidal functions to capture some specific/dominant features of the nonlinear function to be approximated (Pei and Mai, 2008). We match each dominant feature to a prototype before concatenating them within one or more hidden layers for backpropagation to be applied. This technique captures important features and discloses more inner-workings through visualization, theoretical, and mathematical work. This study helps understand, using a meaningful application, what ANN prototypes really are and how to use ANN prototypes effectively, and even make new prototypes.

2.4 Robotic arm and target equations

Robot kinematics may serve as a useful domain for developing the theory for I^3 by providing a scalable application problem. This is because robot kinematics allows us to start simple, then gradually add complexity in the number of input variables, the number of output variables, and the complexity of the relationships between them, thus provides a path for developing more complex ANNs. Increase in complexity of relationships requires more hidden layers.

Robot kinematics can also be useful as a gradually extensible application domain. Using forward kinematic equations, it is possible to describe the idealized location of a robot arm. However, in practice there are problems with joint slippage, sag, and other mechanical issues. When greater weight is carried, the arm tends to sag more from the ideal position, potentially resulting in large discrepancies between what the idealized model predicts and what is observed in practice. As we deal with more complex robotic arms, it becomes more difficult to adequately model them using just equational models. For these reasons, it is appropriate to learn the model by trying different joint angles and measuring the location of the end effector. One approach is to build ML models of robotic arms. This work helps users to understand models and potentially provides better models.

2.4.1 Simplified PUMA robotic arm

The target robot arm is a simple two-link robot arm, as depicted in Fig. 2.8 with waist joint θ_1 that rotates 360 degrees in the horizontal plane; shoulder joint θ_2, that rotates 360 degrees vertically, with 0 degrees straight out to one side; and an elbow joint θ_3 that also rotates vertically, with 0 degrees bent up at a right angle to the first link. The elbow rotation can be from -90 to $+90$ degrees (considered the "elbow forward" configuration) or $+90$ to $+270$ degrees (considered the "elbow backward" configuration). Each link (l_1 and l_2) is 1/2 unit length and the first link (l_1) is situated at the origin. The arm is assumed to have zero volume which allows for full range of motion on all joints without the risk of self collisions.

2.4.2 Target equations

The kinematics equations for the series chain of a robot with revolute or prismatic joints are obtained using a rigid transformation $[J_i]$ to characterize the relative movement allowed at each joint and a separate rigid transformation $[K_i]$ to define the dimensions of each link. The result is a sequence of alternating joint and link transformations from the base to the end link,

$$[T] = [J_1][K_1][J_2][K_2]\dots[J_n][K_n] \tag{2.2}$$

where $[T]$ is the transformation locating the end-link. The equations for the forward kinematics of our simplified arm are obtained using (2.2) and can be seen in Table 2.2.

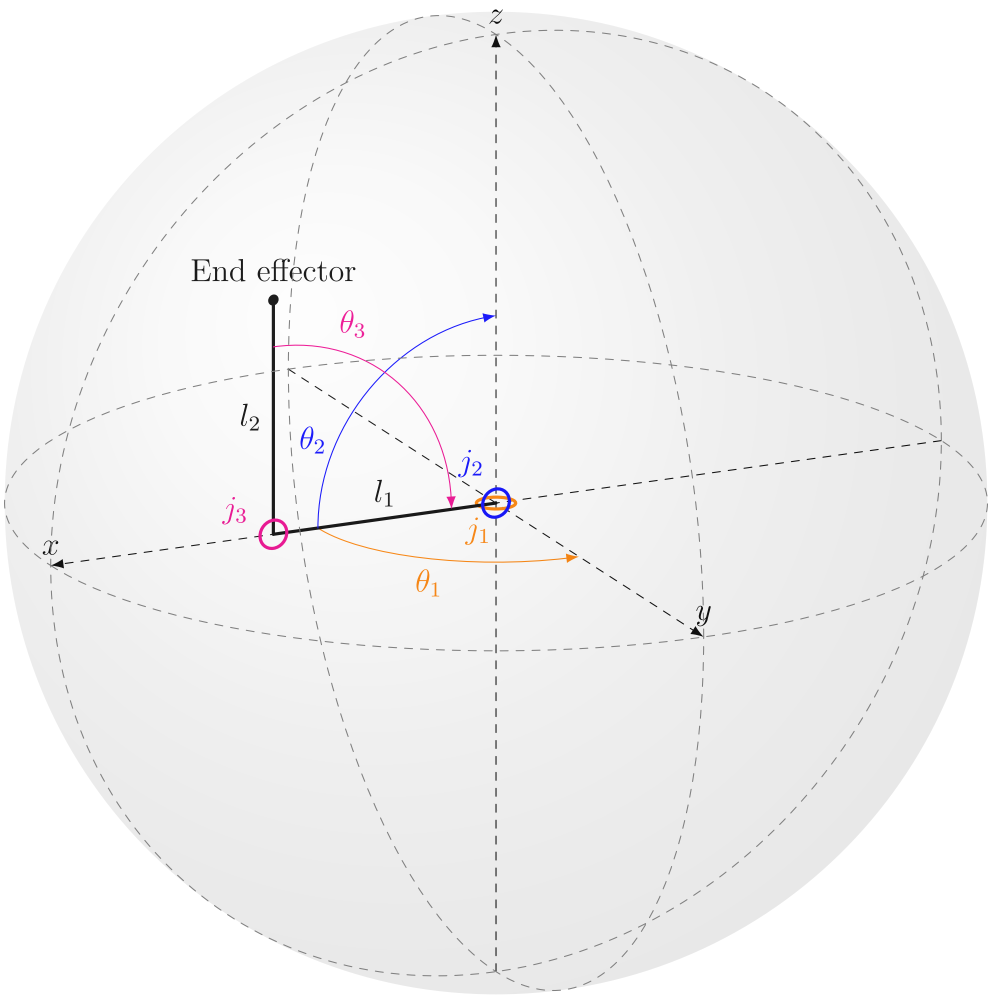


Figure 2.8: 3-D orientation of simplified PUMA robotic arm.

Table 2.2: Equations for different input-output combinations derived using forward kinematics of robotic arm. In the kinematics equations, the joints has only one cycle $(0 - 2\pi)$. *Note:* Some of the input-output combinations are excluded from the table as they either have constant value or have the same as other existing input-output combination.

Inputs	**Output**	**Equation**	
$\hat{\theta}_1$	$\hat{x}$	$0.5\cos(2\pi\hat{\theta}_1) + 0.5$	(2.3)
$\hat{\theta}_1$	$\hat{y}$	$0.5\sin(2\pi\hat{\theta}_1) + 0.5$	(2.4)
$\hat{\theta}_2$	$\hat{x}$	$0.5\cos(2\pi\hat{\theta}_2 + \pi/4) + 0.5$	(2.5)
$\hat{\theta}_2$	$\hat{z}$	$0.5\sin(2\pi\hat{\theta}_2 + \pi/4) + 0.5$	(2.6)
$\hat{\theta}_3$	$\hat{x}$	$-0.5\sin(2\pi\hat{\theta}_3) + 0.5$	(2.7)
$\hat{\theta}_1$, $\hat{\theta}_2$	$\hat{x}$	$0.5[\cos(2\pi\hat{\theta}_1)\cos(2\pi\hat{\theta}_2 + \pi/4) + 1]$	(2.8)
$\hat{\theta}_1$, $\hat{\theta}_2$	$\hat{y}$	$0.5[\sin(2\pi\hat{\theta}_1)\cos(2\pi\hat{\theta}_2 + \pi/4) + 1]$	(2.9)
$\hat{\theta}_2$, $\hat{\theta}_3$	$\hat{x}$	$0.25\cos(2\pi\hat{\theta}_2) - 0.25\sin(2\pi(\hat{\theta}_2 + \hat{\theta}_3)) + 0.5$	(2.10)
$\hat{\theta}_2$, $\hat{\theta}_3$	$\hat{z}$	$0.25\sin(2\pi\hat{\theta}_2) + 0.25\cos(2\pi(\hat{\theta}_2 + \hat{\theta}_3)) + 0.5$	(2.11)
$\hat{\theta}_1$, $\hat{\theta}_3$	$\hat{x}$	$0.25\cos(2\pi\hat{\theta}_1) * [1 - \sin(2\pi\hat{\theta}_3)] + 0.5$	(2.12)
$\hat{\theta}_1$, $\hat{\theta}_3$	$\hat{y}$	$0.5\sin(2\pi\hat{\theta}_1) * \cos(2\pi(\hat{\theta}_3 + 1/8)) + 0.5$	(2.13)
$\hat{\theta}_1$, $\hat{\theta}_2$, $\hat{\theta}_3$	$\hat{x}$	$-0.25\cos(2\pi\hat{\theta}_1) * [\sin(2\pi(\hat{\theta}_2 + \hat{\theta}_3)) - cos(2\pi\hat{\theta}_2)] + 0.5$	(2.14)
$\hat{\theta}_1$, $\hat{\theta}_2$, $\hat{\theta}_3$	$\hat{y}$	$0.5 - 0.25\sin(2\pi\hat{\theta}_1) * [\sin(2\pi(\hat{\theta}_3 - \hat{\theta}_2)) + cos(2\pi\hat{\theta}_2)]$	(2.15)

Chapter 3

Research Questions

This chapter introduces the research questions of this study. The research questions correspond to structural issues, initialization. Below are the list of research questions used in this study.

1. Is the I^3 to structural determination sufficient to adapt to robot kinematics?
2. Is the I^3 to structural determination sufficient to allow for appropriate design of multilayer feedforward ANNs?
3. Is the I^3 to initialization adapt to robot kinematics?
4. While concatenating, does the initial parameters of these prototypes effect their learning ability?
5. For the structure recommended by the I^3, what is the performance of I^3 initialization w.r.t. other state-of-the-art initialization?
6. When compared to other initialization models, are the I^3 models robust to noise?

Research questions related to structural issues will be covered in Section 3.1, initialization issues will be covered in Section 3.2, and other issues will be covered in Section 3.3.

3.1 Structural issues

Research questions in this section focus on structural aspect of I^3. First research question focus on structural transferability of I^3, this includes evaluating the performance of two robot kinematics functions corresponding to two different prototypes. Second research question focus on extensibility of I^3 for sigmoidal ANNs with more hidden layers.

3.1.1 Research question 1.1 - Structural transferability

Motivation: The I^3 has been applied extensively to engineering mechanics (Pei et al., 2021; Pei, 2001; Pei et al., 2005, 2013; Pei and Mai, 2008; Pei et al., 2008) and also to epidemics (Hougen et al., 2020). Transferring this approach to another domain helps to advance theory of I^3 and also advance the applicability of this approach to other domains.

Research question: Is the I^3 to structural determination sufficient to adapt to robot kinematics? First component in adapting to robot kinematics is determining structure for ANNs and these structures give us relatively small architecture that is still sufficient to adequetely represent the function once it is modeled.

Hypothesis 1.1.a: The structure recommended by prototype function in Fig. 11(b) of (Pei et al., 2013) will give sufficient representational power to approximate $\hat{x} = 0.5 + 0.5\cos(2\pi\hat{\theta}_1)$.

Hypothesis 1.1.b: The structure recommended by prototype 3 - variant c in Fig. 3 of (Pei and Mai, 2008) will give sufficient representational power to approximate $\hat{y} = 0.5 + 0.5\sin(2\pi\hat{\theta}_1)$, $\hat{x} = 0.5 - 0.5\sin(2\pi\hat{\theta}_3)$.

Reasoning for hypotheses: The I^3 is about approximating nonlinear static functions thus it has the potential of being applied to where approximating nonlinear static functions is relevant.

3.1.2 Research question 1.2 - Extensibility to more hidden layers

Motivation: Having able to extend the I^3 for multilayer feedforward neural networks with more hidden layers shows the extensibility of this approach. Extensibility allows us to apply this approach to wide variety of problems.

Research question: Is the I^3 to structural determination sufficient to allow for appropriate design of multilayer feedforward ANNs?

For this research question we have three sub hypothesis. First is related to ANNs with two hidden layers, while the second is related to ANNs with three hidden layers and the third is related to ANNs with four hidden layers.

Hypothesis 1.2.a: Following this I^3 technique will lead to neural networks that are able to approximate two hidden layers. The functions that require two hidden layers correspond to Equation 2.5, Equation 2.6, Equation 2.8, Equation 2.9, Equation 2.10, Equation 2.11, and Equation 2.12.

Hypothesis 1.2.b: Following this I^3 technique will lead to neural networks that are able to approximate three hidden layers. The function that require three hidden layers correspond to Equation 2.13.

Hypothesis 1.2.c: Following this I^3 technique will lead to neural networks that are able to approximate four hidden layers. The function that require four hidden layers correspond to Equation 2.14.

Reasoning for hypotheses: The I^3 is applied to approximate functions with one, two hidden layers. Thus, it has the potential to apply for functions requiring more hidden layers; Complex functions can be modeled by concatenating multiple smaller networks.

3.2 Initialization

Research questions in this section focus on initialization aspect of I^3. First research question focus on domain transferability of I^3, this includes evaluating the performance of two robot kinematics functions corresponding to two different prototypes. Second research question focus on concatenating multiple prototypes and the third research question focus on performance evaluation.

3.2.1 Research question 2.1 - Domain transferability

Motivation: The I^3 has been applied extensively to engineering mechanics (Pei et al., 2021; Pei, 2001; Pei et al., 2005, 2013; Pei and Mai, 2008; Pei et al., 2008) and also to epidemics (Hougen et al., 2020). Transferring I^3 to another domain helps to advance theory of this I^3 and also advance the applicability of this approach to other domains.

Research question: Is the I^3 to initialization adapt to robot kinematics? Second component in adapting to robot kinematics is to have initial values that have advantages like having low initial error. We want to determine structure and initial values for couple of representative equations.

Hypothesis 2.1.a: The initial values recommended by prototype function in Fig. 11(b) of (Pei et al., 2013) will facilitate to approximate $\hat{x} = 0.5 +$

$0.5\cos(2\pi\hat{\theta}_1)$.

Hypothesis 2.1.b: The initial values recommended by prototype 3 - variant c in Fig. 3 of (Pei and Mai, 2008) will give sufficient representational power to approximate $\hat{y} = 0.5 + 0.5\sin(2\pi\hat{\theta}_1)$, $\hat{x} = 0.5 - 0.5\sin(2\pi\hat{\theta}_3)$.

Reasoning for hypotheses: The I^3 is about approximating nonlinear static functions thus it has the potential of being applied to where approximating nonlinear static functions is relevant.

3.2.2 Research question 2.2 - Concatenating prototypes

Motivation: Increase in inputs or increase in hidden layers results in concatenating multiple prototypes. When concatenating prototypes, preventing the *exploding gradients problem* (large error gradients results in extremely large updates to ANN model weights during training) helps models for stable learning.

Research question: Concatenating and condensing networks results in multiplication of incoming and outgoing weights. While concatenating, does the initial parameters of these prototypes effect their learning ability?

Hypothesis 2.2.a: Concatenating prototypes results in *exploding gradients* i.e., large error gradients accumulate and leads to large updates in ANNs, which results in unstable learning.

Reasoning: When we concatenate and condense networks, this results in multiplication of incoming and outgoing weights and causes larger weights in the ANN which could lead to exploding gradients.

Hypothesis 2.2.b: Percentage of functions that become unstable increase with increase in number of hidden layers.

Reasoning: Weights explode more with increase in number of hidden layers and this makes model more unstable.

3.2.3 Research question 2.3 - Performance evaluation

For the structure recommended by the I^3, what is the performance of I^3 initialization w.r.t. other state-of-the-art initialization?

Hypothesis 2.3.a: In general, we expect our I^3 to have better performance in terms of error, i.e., lower Mean Squared Error (MSE).

Hypothesis 2.3.b: In general, we expect our I^3 to be more consistent, i.e., does not get stuck at local optima, when compared to Xavier or Nguyen-Widrow initialization.

Reasoning for hypotheses: Initial values of the prototypes are obtained based on theoretical justification and expect the model to start at right part of search space. So, we expect I^3 to be consistent and have better performance.

3.3 Other Issues

3.3.1 Research Question 3.1 - Robustness to noise

Motivation: It is important to design models that also perform well when there are mechanical issues in the robotic arm. Designing such models helps roboticists make practical decisions in real-world applications.

Research question: Are the models developed using I^3 approach robust to noise, i.e., with increase in the noise does the models learn the function over noise? Hypothesis 3.1.a: We believe that increase in noise does not favor any of the initialization approach (NW or I^3).

Reasoning for hypothesis: For noisy data, we do not have any evidence that suggests either of the initialization approach performs better than the other.

Chapter 4

Experiments

An objective of our work is to see the feasibility of designing interpretable ANN models for a simplified robotic arm. Apart from interpretability, we are hypothesizing that ANNs trained using I^3 will have training success and good generalization capabilities. We are also interested analyzing the performance when varying the number of input features. In this work we have used one, two, and three input features to predict a target variable. This helps us analyze the complexities and performance involved while training with more input features. Discussed below are data generation, experimental setup, and initial parameters for the I^3 used in this paper.

4.1 Data generation

Synthetic motor noise is obtained by adding noise after the input. For each input-output combination, we generate data at four different noise levels—no, low, medium, and high noise. Low, medium, and high noise data have zero-mean noise with a standard deviation of 5, 10, and 15 degrees, respectively. Using the equations in Table 2.2, data is generated by adding noise to the input angles before computing the output, resulting in noisy output while retaining the original (non-noisy) value for each input.

The noise in the high noise case is greater than is generally observed in real-world use, although if arms were used nearer to their physical carrying capacity, such noise might be seen. However, noise model used in this research is crude and is not actually a good representation of overall types of errors we would likely have for a real robotic arm. Here, we are also assuming that the robotic arm itself does not occupy any physical space.

In our kinematics we have three input angles (θ_1, θ_2, and θ_3) that determine the end position (x, y, and z) of the robot end effector. When we experiment with one input, we are interested in learning the input-output relation w.r.t. that single input. Then we vary that input alone by keeping all the remaining inputs constant at 0. When we experiment with two inputs, we are interested in learning the output relation w.r.t. these two inputs. Then we vary only those two inputs. While modeling, varying inputs are treated as inputs to a neural network.

For each input-output combination, we have generated 4,000 samples of data for each category of noise. All the input angles are randomly generated covering the entire input range. Based on the noise levels, noise is randomly added to the data. While training each model, 3,000 samples of this data are used for training, 500 samples are used for validation, and the remaining 500 samples are used for testing.

Input angles vary from 0 to 360 degrees (0 to 2π radians) and are normalized between 0 and 1. Normalizing inputs is common practice in ML as it speeds convergence. For every input combination (varying inputs), we obtained possible minimum and maximum values for each output. These minimum and maximum values are used as lower and upper bound to normalize each output variable. By normalizing outputs, error reported will be uniform across all input-output combinations and thus helps for easy and fair comparison. Thus, inputs ($\hat{\theta}_1$, $\hat{\theta}_2$, and $\hat{\theta}_3$) and outputs ($\hat{x}$, $\hat{y}$, and $\hat{z}$) are scaled between 0 and 1. This scaled data is used for training the ANNs.

4.2 Experimental design

To compare with our I^3 initialization, we use NW and Xavier initializations. The Adam optimizer is used for training these ANNs. For shallow neural networks, we have observed that the LM optimizer performs better than Adam. However, we used Adam here as we are interested in continuing on to train the deeper neural networks that will be engineered when we add more inputs. LM is not feasible for training deep neural networks as it is very slow to converge, this is particularly true if the model has more parameters which requires the algorithm to converge slowly. One of the other limitation of LM is, when the least-squares function is very flat, the algorithm may easily become lost in parameter space (Transtrum and Sethna, 2012).

A learning rate of 0.007 was chosen based on preliminary experimentation. We used a mini-batch size of 32 and trained models for $1,500$ epochs. Gradient descent updates weights of ANNs after calculating the error on the whole dataset, on the other hand stochastic gradient descent updates weight after calculating the error on a single observation. Gradient descent is is slow but stable, on the other hand stochastic gradient descent is fast but unstable. Mini-batch gradient descent updates weights after calculating error on certain/less number of observations;

This results in fast and stable learning. Usually it is preferred to use the size of mini-batch in the powers of 2 (16, 32, 64, 128); this helps for better use of available computational resources as computers inherently use binary code. Models are trained for 1500 epochs as we observed that it takes 1500 epochs to learn and converge to a solution. When MSE is used as the loss function, the optimizer tries to reduce the square of the error, which magnifies the penalties of larger errors as compared to those of smaller errors. Based on the applications of robotic arms, it makes sense to more substantially penalize larger errors. Thus, the loss and the evaluation metric used are both MSE.

For each dataset, we do 30 repetitions to collect statistically meaningful results. For each repetition, data is randomly split between training, validation, and test. Models are trained with initial parameters and their performance is recorded.

To compare results, the Mann–Whitney U test is chosen because it is a non-parametric test used to compare two distributions of independent samples. Non-parametric methods allow statistical inferences without making the assumption that the samples have been taken from a particular distribution. The Mann–Whitney U test is a commonly used alternative to two sample t-test when the distribution is non-normal (McKnight and Najab, 2010).

4.3 Prototypes used

Not all prototypes to be presented are designed in this study. In fact, only a small number is created in this study. In this work the author have not only used prototypes but also learned why prototypes work, and how to use these prototypes to new target equations; More impor-

tantly using the *function composition*, i.e., building larger functions from smaller functions, along with multiple prototypes. Where each prototype is used to build a smaller function.

A *prototype* is a design of often small sigmoidal ANN with one hidden layer to approximate a particular mathematical operation, which is often normalized in terms of input(s) and/or output. Not all situations for applying the existing prototypes are defined rigorously. As a result, some prototypes have variants, e.g., Prototypes 1 to 3, each with three variants

When a prototype is used to a specific target function (or a mathematical operator), one has to pay attention to the input range(s) and output range so that a chosen prototype can be applied. The resulted application of a prototype can be considered as an application instance of the chosen prototype. This is a design process, meaning that multiple choices could be possible.

In the study, the author took a significant amount of time to learn the prototypes relevant to this study, and then applied them correctly and as effectively as possible. However, different choices of prototypes, prototype variants (if applicable), and instances are possible. The author mostly limited his choice to just one to follow the defined research questions.

Prototypes related to approximating constant and polynomial operations are discussed in (Pei et al., 2005) and prototypes related to approximating four basic arithmetic operations are discussed in (Pei et al., 2013). The prototpye used for cos is discussed in (Pei et al., 2013). Below are the three prototypes that are used to approximate equations.

4.3.1 Approximating constant - Prototype A

Prototypes related to approximating constant is discussed in (Pei et al., 2005). Approximating constant is explained in Section 2.3. Initial parameters for approximating a constant are in Table 2.1. These values can be used to approximate any constant. Below are some of the subprototypes used to approximate the constant.

1. Approximating 1 - Instance A.1:

   This requires two hidden nodes. W_1 are $[1, -1]$, and W_2 has the value of constant we would like to approximate, so W_2 becomes $[1, 1]$, b_1 takes the values of $[0, 0]$, and b_2 equals $[0]$.

2. Approximating 0.5 - Instance A.2:

   This requires two hidden nodes. W_1 are $[1, -1]$, and W_2 has the value of constant we would like to approximate, so W_2 becomes $[0.5, 0.5]$, b_1 takes the values of $[0, 0]$, and b_2 equals $[0]$.

3. Approximating 0.125 - Instance A.3:

   This requires two hidden nodes. W_1 are $[1, -1]$, and W_2 has the value of constant we would like to approximate, so W_2 becomes $[0.125, 0.125]$, b_1 takes the values of $[0, 0]$, and b_2 equals $[0]$.

4.3.2 Approximating arithmetic and polynomial operations - Prototype B

Prototypes related to approximating polynomial operations are discussed in (Pei et al., 2005) and prototypes related to approximating four basic arithmetic operations are discussed in (Pei et al., 2013). Initial parameters for approximating arithmetic and polynomial operations are in Table 2.1. Below are some of the subprototypes used to approximate arithmetic and polynomial operations.

1. Approximating first order polynomial (p^1) - Instance B.1:

   This requires two hidden nodes. W_1 are $[0.1, -0.1]$, and W_2 are $[20, -20]$, b_1 takes the values of $[0, 0]$, and b_2 equals $[0]$.

2. Approximating addition of two first order polynomials ($p_1 + p_2$) - Instance B.2:

   This requires four hidden nodes. W_1 are $[[0.1, -0.1, 0, 0], [0, 0, 0.1, -0.1]]$ ([0.1, -0.1, 0, 0] corresponds to incoming weights from p_1 and [0 , 0, 0.1, -0.1] corresponds to incoming weights from p_2), and W_2 are $[20, -20, 20, -20]$, b_1 takes the values of $[0, 0, 0, 0]$, and b_2 equals $[0]$.

3. Approximating multiplication of two first order polynomials ($p_1 * p_2$) - Instance B.3:

   This requires eight hidden nodes. W_1 are
   $[[0.1, -0.1, 1, -1, 0.1, -0.1, 1, -1], [0.1, -0.1, -1, 1, -0.1, 0.1, -1, 1]]$ and
   W_2 are [5.507e5, 5.507e5, -50.006, -50.006, -5.507e5, -5.507e5, 50.006, 50.006], b_1 takes the values of $[-10, -10, 0, 0, -10, -10, 0, 0]$, and b_2 equals $[0]$.

4.3.3 Approximating cosine curves - Prototype C

$\hat{\theta}$ is an input angle and it varies from 0 to 1.

1. Approximating $0.5 \cos(2\pi \times t)$ - Instance C.1:

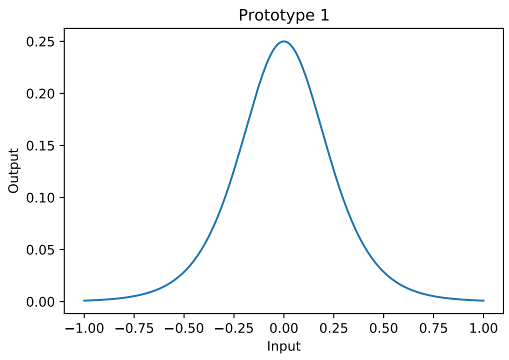


Figure 4.1: Prototype C used to approximate cosine term.

The prototpye used for cos is discussed in (Pei et al., 2013). Fig. 4.2(a) shows the function to be approximated. Dominant features including visually observable features in the data help us to design the architecture and initial values of neural network. This function closely takes the shape of the prototype function in Fig. 11(b) of (Pei et al., 2013), which is also depicted in Fig. 4.1 where the first order partial derivative of a sigmoidal function (S) with respect to bias is observed

$$\frac{\partial S(W,p,b)}{\partial b} = \frac{e^{-(Wp+b)}}{(1+e^{-(Wp+b)})^2} \tag{4.1}$$

and is plotted using $W = 7$ and $b = 0$, where W and b are the incoming weight and bias of the neuron, and p is the input. This prototype function in Equation (4.1) can be approximated using linear sum of two sigmoidal functions (Hougen et al., 2020) and the equation is as follows

$$\frac{\partial S(W,p,b)}{\partial b} \approx \frac{1}{2\Delta b}\big(S(W,p,b+\Delta b) - S(W,p,b-\Delta b)\big) \tag{4.2}$$

Equation (4.2) is Euler forward formula and can be approximated using two hidden nodes, as shown in Fig. 2.6. We choose a small value of Δb equals to 0.1, it is an empirical choice and needs to be a small value. We are doing an approximation of $\frac{\partial S(W,p,b)}{\partial b}$ .Based on the above information, W_1 are [7, 7]. From (4.2), we can observe that W_2 are $[\frac{1}{2\Delta b}, -\frac{1}{2\Delta b}]$ respectively. Thus, W_2 becomes [5, -5]. b_1 takes the values of $[b + \Delta b,\ b - \Delta b]$, b_1 equals [0.1, -0,1], and b_2 equals [0].

Initial weights to approximate are obtained in three steps. First we scale the prototype function to our required input and output range. Our required input and output range is from 0 to 1 and can be observed in Fig. 4.2(a). The input and output range for the prototype is from -1 to 1 and 0 to 0.25 respectively, which can be seen in Fig. 4.1. The prototype function can be made closer to Fig. 4.2(a), by only considering the input range between [-0.6 to 0.6], the weights and bias of scaled function can be obtained by substituting values of $\hat{\theta}_1 = 0$ for p = -0.6 and $\hat{\theta}_1 = 0.5$ for $p = 0$. Now, W_1 changes to [8.4, 8.4] and b_1 changes to [-4.1, -4.3]. To scale the output between 0 and 1, W_2 are multiplied by 4, thus W_2 changes to [20, -20]. Now these weights are better used for our input and output ranges.

In second step, we flip the function by changing the sign of the output. Thereby bringing the prototype function more closer to function in Fig. 4.2(a). Sign of the output can be reversed by changing the sign of the final layer weights, thus W_2 becomes [-20, 20].

In final step, we add a constant of 1, to bring this function more closer to the function in Fig. 4.2(a). To add a constant we need two hidden nodes as described in Chapter 2, Section 2.3. Now W_1 corresponding to these two hidden nodes are [1, -1], b_1 corresponding to these two hidden nodes are [0, 0], and W_2 corresponding to these two hidden nodes are [1, 1]. Now, we have a total of 4 hidden nodes, 2 from earlier prototype function and 2 from the the addition of constant. Thus our final initial parameters are: W_1 are [8.4, 8.4, 1, -1], b_1 are [-4.1, -4.3, 0, 0], W_2 are [-20, 20, 1, 1], and b_2 equals [0]. Now these weights are used to approximate function in Fig. 4.2(b). This function is very close to the function we are interested in approximating, i.e., Fig. 4.2(a). These parameters takes the shape of ANN architecture in Fig. 4.3.

4.4 Initial architecture and parameter values for $\mathbf{I}^3$

Equations in Table 2.2, do not include constant values; approximating a constant does not require any nonlinear approximation and can be approximated using two hidden nodes as described in Prototype A.

In this section, detailed explanations will be given first to how to apply prototypes for typical target functions and also demonstrates the architecture and getting initial values using three equations. Architecture and initial values for other equations are explained in Section A.1 of this work.

When the complexity of a target function goes up, function composition is applied to identify first individual terms to be approximated, and then how each individual term can be composed step-by-step into mathematical operators. ANN prototypes are then applied step-by-step to approximate these identified mathematical operators, aiming for constructing one neural network for the target function in an interpretable manner.

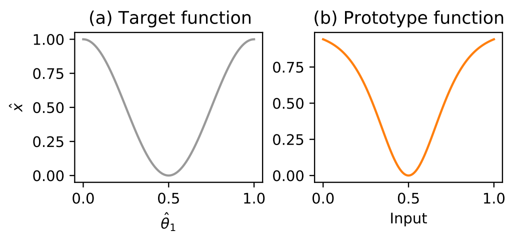


Figure 4.2: $\hat{x}$ to be approximated given $\hat{\theta}_1$ (a) Target function to be approximated (b) Prototype function obtained using I^3 to approximate target function.

The number of the individual terms making up the target function is the number of sub-neural networks. The number of these steps for an individual term is often the number of hidden layers requires for the sub-neural network. Since different individual terms making up a function can have different complexity, the resulted sub-neural network can demand different numbers of hidden layers. This is a challenge to constructing a standard, fully connected *feedforward neural network*. Feedforward neural network is a class of ANNs in which the connections between nodes do not form a cycle.

To meet this challenge, one of design choices is to identify the deepest sub-neural network, and extend all other sub-neural networks to have the same depth. Extension can be done in different ways, in this study typical treatments are as follows following precious work (Pei et al., 2013; Pei and Mai, 2008). Approximate an input or intermediate output as itself to add one more hidden layer, and the two hidden nodes required to approximate a constant term are only added to the last hidden layer for the target function.

4.4.1 Approximating Equation 2.3: $\hat{x} = 0.5\cos(2\pi\hat{\theta}_1) + 0.5$

Fig. 4.2(a) shows the function to be approximated. Dominant features including visually observable features in the data help us to design the architecture and initial values of neural network. This function closely takes the shape of the prototype function in Fig. 11(b) of (Pei et al., 2013), which is also depicted in Fig. 4.1 where the first order partial derivative of a sigmoidal function (S) with respect to bias is observed in Equation (4.1) and is plotted using $W = 7$ and $b = 0$, where W and b are the incoming weight and bias of the neuron, and p is the input. This prototype function in Equation (4.1) can be approximated using linear sum of two sigmoidal functions (Hougen et al., 2020) as in Equation (4.2).

Equation (4.2) is Euler forward formula and can be approximated using two hidden nodes, as shown in Fig. 2.6. We choose a small value of Δb equals to 0.1, it is an empirical choice and needs to be a small value. We are doing an approximation of $\frac{\partial S(W,p,b)}{\partial b}$ .Based on the above information, W_1 are [7, 7]. From (4.2), we can observe that W_2 are $[\frac{1}{2\Delta b}, -\frac{1}{2\Delta b}]$ respectively. Thus, W_2 becomes [5, -5]. b_1 takes the values of $[b + \Delta b, b - \Delta b]$, b_1 equals [0.1, -0,1], and b_2 equals [0].

Initial weights to approximate are obtained in three steps. First we scale the prototype function to our required input and output range. Our required input and output range is from 0 to 1 and can be observed in Fig. 4.2(a). The input and output range for the prototype is from -1 to 1 and 0 to 0.25 respectively, which can be seen in Fig. 4.1. The prototype function can be made closer to Fig. 4.2(a), by only considering the input range between [-0.6 to 0.6], the weights and bias of scaled function can be obtained by substituting values of $\hat{\theta}_1 = 0$ for p = -0.6 and $\hat{\theta}_1 = 0.5$ for $p = 0$. Now, W_1 changes to [8.4, 8.4] and b_1 changes to [-4.1, -4.3]. To scale the output between 0 and 1, W_2 are multiplied by 4, thus W_2 changes to [20, -20]. Now these weights are better used for our input and output ranges.

In second step, we flip the function by changing the sign of the output. Thereby bringing the prototype function more closer to function in Fig. 4.2(a).

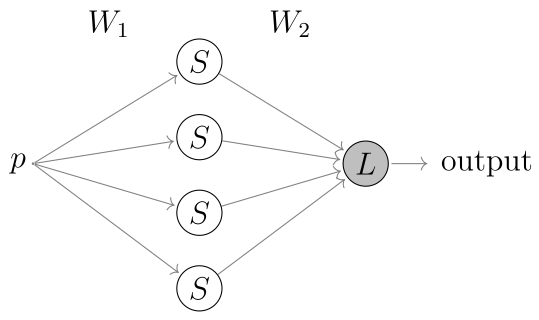


Figure 4.3: ANN architecture with one hidden layer and four hidden nodes. S, L nodes correspond to sigmoidal, linear activation functions respectively.

Sign of the output can be reversed by changing the sign of the final layer weights, thus W_2 becomes [-20, 20].

In final step, we add a constant of 1, to bring this function more closer to the function in Fig. 4.2(a). To add a constant we need two hidden nodes as described in Chapter 2, Section 2.3. Now W_1 corresponding to these two hidden nodes are [1, -1], b_1 corresponding to these two hidden nodes are [0, 0], and W_2 corresponding to these two hidden nodes are [1, 1]. Now, we have a total of 4 hidden nodes, 2 from earlier prototype function and 2 from the the addition of constant. Thus our final initial parameters are: W_1 are [8.4, 8.4, 1, -1], b_1 are [-4.1, -4.3, 0, 0], W_2 are [-20, 20, 1, 1], and b_2 equals [0]. Now these weights are used to approximate function in Fig. 4.2(b). This function is very close to the function we are interested in approximating, i.e., Fig. 4.2(a). These parameters takes the shape of ANN architecture in Fig. 4.3.

Note: For the later functions involving cosine terms, we will be using the above mentioned prototype with some variation in weights and bias that suits well with the function to be approximated. From here on, this prototype will be referenced as “Prototype C”.

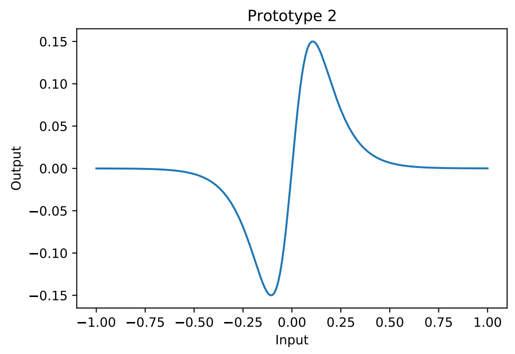


Figure 4.4: Prototype 2 used to approximate sine term.

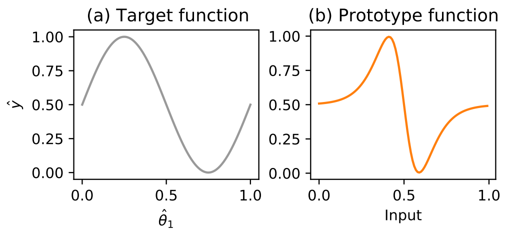


Figure 4.5: $\hat{y}$ to be approximated given $\hat{\theta}_1$ (a) Target function to be approximated (b) Prototype function obtained using I^3 to approximate target function.

4.4.2 Approximating Equation 2.4: $\hat{y} = 0.5\sin(2\pi\hat{\theta}_1) + 0.5$

Fig. 4.5(a) shows the function to be approximated. This function closely takes the shape of the prototype 3, variant c in Fig. 3 of (Pei and Mai, 2008), which is also depicted in Fig. 4.4. To approximate this prototype function we require two hidden nodes, similar to Fig. 2.6. The initial parameters to approximate this function follows, W_1 are [20, 10], W_2 are [1, -1]. b_1 takes the values of [0, 0] and b_2 equals to [0]. These initial values can be found in Table. 1 of (Pei and Mai, 2008).

Similar to equation in Subsection 4.4.1.a, initial weights to approximate are obtained in three steps. First we scale the prototype function close to our required input and output range. Our required input and output range is from 0 to 1.

The input and output range for the prototype is from -1 to 1 and -0.15 to 0.15 respectively. The prototype function can be made closer to Fig. 4.5(a), by only considering the input range between [-0.6 to 0.6]; the weights and bias of scaled function can be obtained by substituting values of $\hat{\theta}_1 = 0$ for $p =$ -0.6 and $\hat{\theta}_1 =$ 0.5 for $p = 0$. Now, W_1 changes to [24, 12] and b_1 changes to [-12, -6]. To scale the output between -0.5 and 0.5 (the difference between minimum and maximum value of function changes from 0.3 to 1.0), W_2 are multiplied by 3.3, thus W_2 changes to [3.3, -3.3]. Now these weights are better used for our input and output ranges.

In second step, we flip the function by changing the sign of the output. Thereby bringing the prototype function more closer to function in Fig. 4.5(a). Thus W_2 becomes [-3.3, 3.3]. In final step, we add a constant of 0.5, this changes output range from -0.5 to 0.5 to our required range of 0 to 1. To add a constant we need two hidden nodes. Now W_1 corresponding to these two hidden nodes are [1, -1], b_1 corresponding to these two hidden nodes are [0, 0], and W_2 corresponding to these two hidden nodes are [0.5, 0.5]. Now, we have a total of 4 hidden nodes, 2 from earlier prototype function and 2 from the the addition of a constant. Thus our final initial parameters are: W_1 are [24, 12, 1, -1], b_1 are [-12, -6, 0, 0], W_2 are [-3.3, 3.3, 0.5, 0.5], and $b_2 = [0]$. Now these weights are used to approximate function in Fig. 4.5(b). This function is very close to the function we are interested in approximating, i.e., Fig. 4.5(a). These parameters takes the shape of ANN architecture in Fig. 4.3.

Note: For the later functions involving sine terms, we will be using the above mentioned prototype with some variation in weights and bias that suits well with the function to be approximated. From here on, this prototype will be referenced as “Prototype 2”.

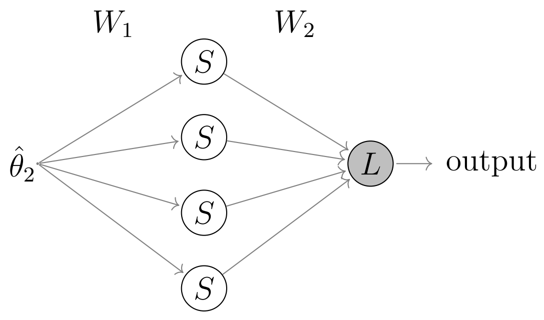


Figure 4.6: ANN architecture to approximate term 1. S, L nodes correspond to sigmoidal, linear activation functions respectively.

4.4.3 Approximating Equation 2.10

$$\hat{x} = \underbrace{0.25\cos\left(2\pi\hat{\theta}_2\right)}_{\text{Term 1}} + \underbrace{\left[-0.25\sin\left(2\pi\left(\hat{\theta}_2+\hat{\theta}_3\right)\right)\right]}_{\text{Term 2}} + \underbrace{0.5}_{\text{Term 3}} \tag{4.3}$$

This equation can be approximated by first approximating three of its components separately, then using these approximations to approximate the whole equation. First part involves approximating $0.25\cos(2\pi\hat{\theta}_2)$, second part involves approximating $-0.25\sin(2\pi(\hat{\theta}_2+\hat{\theta}_3))$ and the later part involves approximating 0.5.

Approximating Term 1: $0.25\cos(2\pi\hat{\theta}_2)$

Four hidden nodes are used to approximate this term, i.e, $0.25\cos(2\pi\hat{\theta}_2)$ given $\hat{\theta}_2$ as input. The initial weights and bias for this component are obtained similar to Subsection 4.4.1. Thus, W_1 becomes to [8.4, 8.4, 1, -1] and bias of hidden neurons becomes [-4.1, -4.3, 0, 0], W_2 becomes to [-10, 10, 0.25, 0.25], and bias for output linear node is [0]. The architecture to approximate this term is seen in Fig. 4.6.

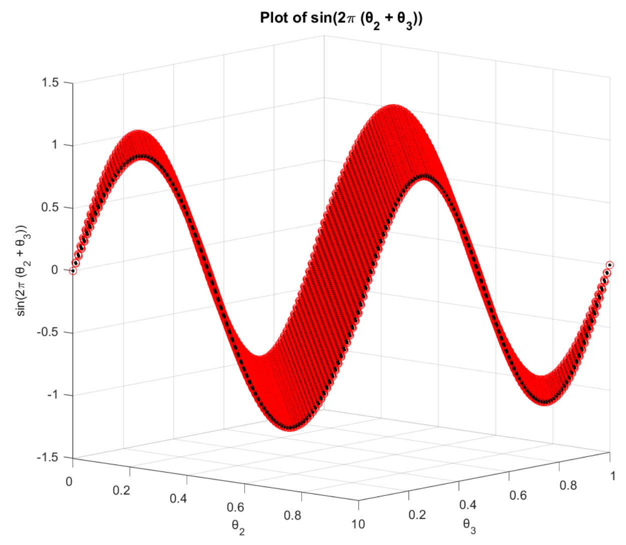


Figure 4.7: Black is for target function, while red is the approximated function (without training).

Approximating Term 2: $-0.25\sin(2\pi(\hat{\theta}_2+\hat{\theta}_3))$

Four hidden nodes are used to approximate $(\hat{\theta}_2+\hat{\theta}_3)$, as it requires four nodes to approximate sum of two first order polynomial terms as mentioned in Table 2.1. So, W_1 are [[0.1, -0.1, 0, 0], [0, 0, 0.1, -0.1]], b_1 are [0, 0, 0, 0], and W_2 are [20, -20, 20, -20] and b_2 is [0].

All the components in earlier equations are designed to take input within the range of 0 to 1 and has one cycle. However, $(\hat{\theta}_2+\hat{\theta}_3)$ has an input range of 0 to 2. $\sin(2\pi(\hat{\theta}_2+\hat{\theta}_3))$ and has two cycles. To facilitate two cycles there are two modifications done while designing the architecture:

1. **For this particular equation, to approximate $\sin$ term we have used prototype 3, variant a in Fig. 3 of (Pei and Mai, 2008) rather than prototype 3, variant c. Variant a is more suitable**

for this equation as output of variant a is from zero to zero so that when we concatenate two of these prototypes side by side, the inference with each other's output is nominal in the summed output.

2. **Two hidden nodes are sufficient to approximate the equation with one cycle -** $-0.25\sin(2\pi(\hat{\theta}_2))$ **or** $-0.25\sin(2\pi(\hat{\theta}_3))$**, as the input range is from 0 to 1. However, to facilitate for an input range of 0 to 2 (two cycles); we need to use the prototype twice which involves some dialations and shifts. Thus, we require two hidden nodes to approximate between 0 to 1 and two additional hidden nodes to approximate between 1 and 2.** Figure 4.7 shows how close is $\sin(2\pi(\hat{\theta}_2 + \hat{\theta}_3))$ to the approximation made using initial values from the above prototype.

Four hidden nodes are used to approximate term 2, i.e, $-0.25\sin(2\pi(\hat{\theta}_2+\hat{\theta}_3))$ given $(\hat{\theta}_2 + \hat{\theta}_3)$ as input. This function $-0.25\sin(2\pi(\hat{\theta}_2 + \hat{\theta}_3))$ closely takes the shape of the prototype 3, variant a. To approximate this prototype function we require two hidden nodes, similar to Fig. 2.6. The initial parameters to approximate this function follows, incoming weights are [10, 5], outgoing weights are [1, -1]. Bias of hidden layer takes the values of [0, 0] and bias at final layer equals to [0]. These initial values can be found in Table. 1 of (Pei and Mai, 2008).

This paragraph explains the initial parameters of first two hidden nodes used to approximate for an input range of 0 to 1. These parameters are obtained in two steps. First we scale the prototype function close to our required input and output range. Our required input range is from 0 to 1 and our required output range is from -0.25 to 0.25. The input and output range for the prototype is from -1 to 1 and -0.15 to 0.15 respectively. The entire input range ([-1.0 to 1.0]) of prototype function can be used to approximate this function; the weights and bias of scaled function can be obtained by substituting values of $\hat{\theta}_1 + \hat{\theta}_2 = 0.5$ for $p = 0$ and $\hat{\theta}_1 + \hat{\theta}_2 = 1$ for $p = 1$. Now, incoming weights changes to [20, 10] and bias of hidden neurons changes to [-10, -5]. To scale the output between -0.25 and 0.25 (the difference between minimum and maximum value of function changes from 0.3 to 0.5), thus the outgoing weights are multiplied by 1.65, thus outgoing weights changes to [1.65, -1.65] and bias for output linear node is [0]. Now these weights are better used for our input and output ranges. In second step, we flip the function by changing the sign of the output. Thereby bringing the prototype function more closer to the function. Thus outgoing weights becomes [-1.65, 1.65].

The initial parameters of next two hidden nodes used to approximate for an input range of 1 to 2. These are obtained in a similar way as above two nodes. The only difference comes in scaling; Here, the weights and bias of scaled function can be obtained by substituting values of $\hat{\theta}_1 + \hat{\theta}_2 = 1.5$ for $p = 0$ and $\hat{\theta}_1 + \hat{\theta}_2 = 1$ for $p =$ -1. Thus, incoming weights changes to [20, 10] and bias of hidden neurons changes to [-30, -15], outgoing weights changes to [-1.65, 1.65], and bias for output linear node is [0]. Thus W_3 is [20, 10, 20, 10], W_4 is [-1.65, 1.65, -1.65, 1.65], b_3 becomes [-10, -5, -30, -15], and b_4 is [0]. The architecture to approximate term 2 is seen in Fig. 4.8 and can be condensed to Fig. 4.9 as explained in (Pei et al., 2013).

Approximating Term 3: 0.5

Approximating 0.5 can be seen in Instance A.2 of Subsection 4.3.1. This requires two hidden nodes. W_1 are [1, -1], and W_2 has the value of constant we would like to approximate, so W2 becomes [0.5, 0.5], b_1 takes the values of [0, 0], and

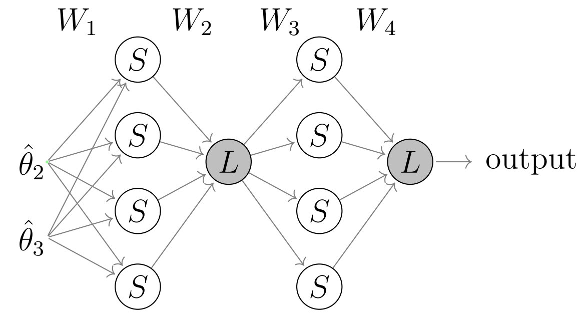


Figure 4.8: Non condensed ANN architecture to approximate term 2. S, L nodes correspond to sigmoidal, linear activation functions respectively.

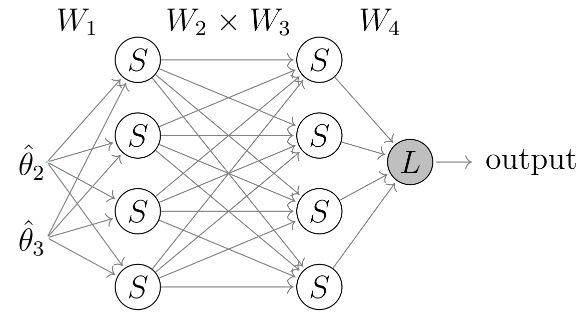


Figure 4.9: Condensed ANN architecture to approximate term 2. S, L nodes correspond to sigmoidal, linear activation functions respectively.

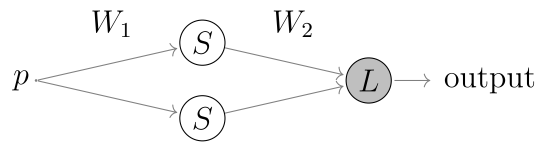


Figure 4.10: ANN architecture to approximate term 3. Nodes labeled S and L have sigmoidal and linear activation functions, respectively and p can be any input/value

b_2 equals [0]. The architecture to approximate term 2 is seen in Fig. 4.9.

Coming to the function composition. Approximating term 1 requires one hidden layer, where as approximating term 2 requires one more additional layer. We want to have a fully connected neural network and term 2 controls the depth of the neural network. So, we use two additional nodes in first hidden layer that takes $\hat{\theta}_2$ as input and gives $\hat{\theta}_2$ as output; this uses the weights from Instance B.1. Term 3 can be added by adding two hidden nodes in the last hidden layer (as approximating constant can take any input value). Fig. 4.11 shows the architecture used to approximate Equation Equation 2.10. This architecture can be condensed to Fig. 4.12 as explained in (Pei et al., 2013).

4.5 Other architectures used

This section shows the architectures of ANNs with three and four hidden layers used to approximate Equation 2.13 and Equation 2.14 respectively.

4.5.1 Architecture to approximate Equation 2.13

$$\hat{y} = 0.5\sin(2\pi\hat{\theta}_1) * \cos(2\pi(\hat{\theta}_3 + 1/8)) + 0.5$$

Approximating above equation requires three hidden layers with 6, 6, 10 units in each of its hidden layer. The condensed architecture can be in seen in Fig 4.13.

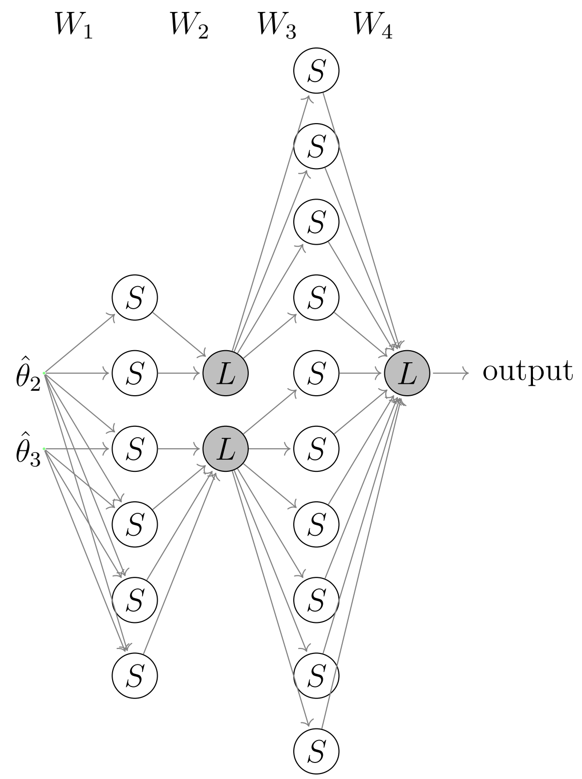


Figure 4.11: ANN architecture before condensing hidden layer - for $\hat{\theta}_2, \hat{\theta}_3$ as inputs and $\hat{x}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively.

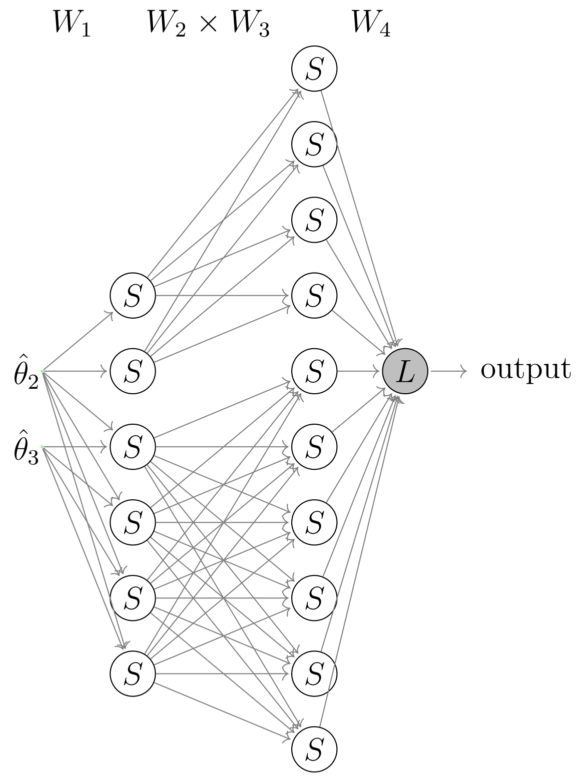


Figure 4.12: ANN architecture after condensing hidden layer - for $\hat{\theta}_2, \hat{\theta}_3$ as inputs and $\hat{x}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively.

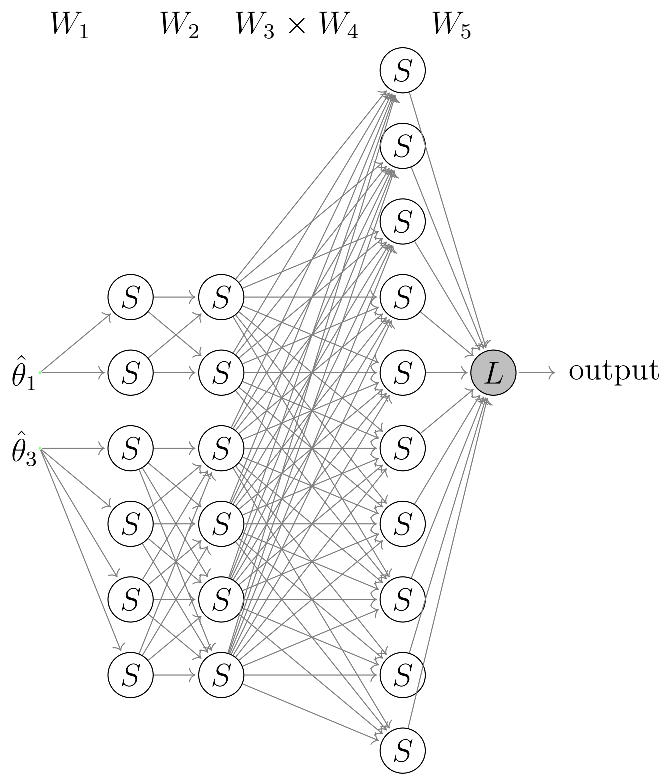


Figure 4.13: ANN architecture after condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_3$ as inputs and $\hat{y}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively.

4.5.2 Architecture to approximate Equation 2.14

$$\hat{x} = -0.25\cos(2\pi\hat{\theta}_1) * [\sin(2\pi(\hat{\theta}_2 + \hat{\theta}_3)) - cos(2\pi\hat{\theta}_2)] + 0.5$$

Approximating above equation requires four hidden layers with 8, 10, 8, 10 units in each of its hidden layer. The condensed architecture can be in seen in Fig 4.14.

4.6 Tools and software used

All our experiments are executed in TensorFlow version 2.5.0 using Python version 3.7.10. Initial weights of neural networks with NW approach are obtained from MATLAB version 9.6. Other details of optimizer, and other algorithms used are mentioned in Section 4.2.

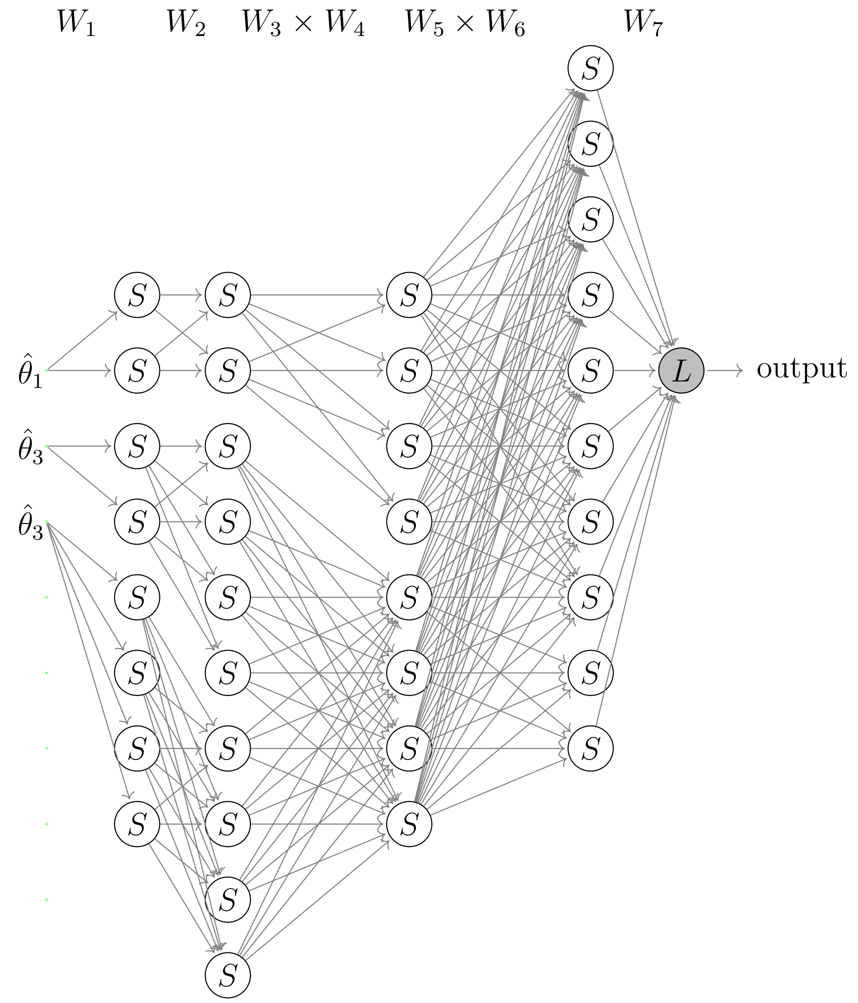


Figure 4.14: ANN architecture after condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3$ as inputs and $\hat{x}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively.

Chapter 5

Results

Results in Table 5.1 and Table 5.2 are obtained when models are trained on training data and evaluated on test data. In Table 5.1, "Best Error" is the smallest error reported in any repetition from any of the initialization approach using the architecture suggested by the I^3 approach. So, values in "Best Error" suggests that the architectures recommended by I^3 approach is capable of approximating the equations; this helps to evaluate research questions related to structural issues (structural transferability, extensibility to more hidden layers).

"Mean MSE" is the mean of the MSE error reported for all the 30 repetitions. Similarly, "Median MSE" is the median of the MSE error reported for all the 30 repetitions. "NW", "Xavier", and "I^3" are the three different initialization approaches used for comparison. For each input output combination, initialization approach with the lowest mean, median error is underlined. M-W $p - value_{NW}$ is the p-value of the Mann-Whitney U test for NW and I^3. M-W $p - value_X$ is the p-value of the Mann-Whitney U test for Xavier and I^3. p-value less than 0.05 indicates the results (errors) are statistically significant and these values are

Table 5.1: Comparison of models with two different initializations. For Noise level, N, L, M, and H correspond to no, low, medium, and high noise, respectively. Best Error is the smallest error reported in any repetition from either approach based on the architecture suggested by the I^3. M-W p-value is the value of the Mann-Whitney U test.

Input	Output	Best Error	M-W		Mean MSE			Median MSE			Repetition Count		
			$p-value_{NW}$	$p-value_X$	NW	Xavier	I^3	NW	Xavier	I^3	NW Local Optima	Xavier Local Optima	I^3 Local Optima
$\hat{\theta}_1$	$\hat{x}$	7.964e-07	*3.019e-11*	*4.615e-10*	5.042e-06	1.779e-05	5.085e-05	4.330e-06	1.710e-05	4.299e-05	0	0	0
	$\hat{y}$	2.052e-07	*1.336e-07*	0.057	2.018e-03	7.290e-06	4.604-06	6.180e-06	4.493e-6	1.849e-06	7	0	0
$\hat{\theta}_2$	$\hat{x}$	1.232e-07	*5.572e-10*	*8.152e-11*	1.891e-05	3.091e-06	6.866e-05	9.755e-07	9.180e-7	5.219e-05	1	0	0
	$\hat{z}$	1.012e-07	*0.009*	0.428	1.382e-05	4.948e-06	8.440e-06	1.500e-06	2.293e-6	3.016e-06	1	0	0
$\hat{\theta}_3$	$\hat{x}$	2.728e-07	*3.988e-04*	0.105	8.171e-04	8.733e-06	5.566e-06	5.789e-06	6.564e-06	2.887e-06	2	0	0
$\hat{\theta}_1, \hat{\theta}_2$	$\hat{x}$	2.650e-05	*1.247e-04*	0.706	2.058e-04	5.126e-04	2.947e-04	1.004e-04	3.513e-04	2.672e-04	0	0	0
	$\hat{y}$	3.558e-05	*2.033e-09*	*5.092e-08*	2.192e-04	3.740e-04	1.007-e03	1.219e-04	2.935e-04	8.345e-04	0	0	0
$\hat{\theta}_2, \hat{\theta}_3$	$\hat{x}$	3.381e-06	*4.077e-11*	*3.019e-11*	3.348e-05	2.574e-05	5.399e-04	2.181e-05	1.806e-05	4.971e-04	0	0	0
	$\hat{z}$	3.211e-06	*3.019e-11*	*3.689e-11*	2.134e-05	6.577e-05	1.696-e03	1.607e-05	2.789e-05	1.006e-03	0	0	0
$\hat{\theta}_1, \hat{\theta}_3$	$\hat{x}$	3.459e-06	*3.019e-11*	*3.019e-11*	2.462e-05	1.952e-05	9.479e-04	1.824e-05	1.480e-05	5.328e-04	0	0	0
	$\hat{y}$	4.668e-06	*3.019e-11*	*3.019e-11*	2.861e-05	4.125e-05	1.059-e02	2.081e-05	3.327e-05	2.050e-03	0	0	0
$\hat{\theta}_1, \hat{\theta}_2, \hat{\theta}_3$	$\hat{x}$	6.619e-05	*3.019e-11*	*5.572e-10*	2.697e-04	1.229e-03	5.588e-03	2.416e-04	1.912e-04	3.222e-03	0	1	0

italicized in the table. Last three columns in Table 5.1 indicates in how many repetitions (out of 30) does NW, Xavier, and I^3 initialization stuck at local optima. All these columns are used to analyze research question - performance evaluation.

In Table 5.2, “Noise level” indicates the level of noise in the data; N, L, M, and H correspond to no, low, medium, and high noise, respectively. “Mean MSE” and “Median MSE” are the mean and median of the MSE error reported across all the 30 repetitions for NW and I^3. “M-W $p - value_{NW}$ is the p-value of the Mann-Whitney U test for NW and I^3” is the p-value of the two sided Mann-Whitney U test for errors reported by NW and I^3. Similar to Table 5.1, p-values are italicized, mean and median errors are underlined. So, Table 5.2 is used to analyze research question - robust to noise.

Fig. 5.1(a), Fig. 5.1(b), and Fig. 5.1(c) are prototypical examples of learning excellently, learning moderately and learning poorly respectively. Actual values in Fig. 5.1 are the values obtained using equations and these values correspond to 0 error. In Fig. 5.1(a), models from both the approaches learned very well to approximate the function over the entire region of input. In Fig. 5.1(b), models from both the approaches moderately learned to approximate function. We can observe that these models did not precisely learn for some input range. NW initialization approach in Fig. 5.1(c) has learned poorly. This is the result of the optimizer stuck at local optima. These figures correspond to training on no noise data (Kanneganti et al., 2021).

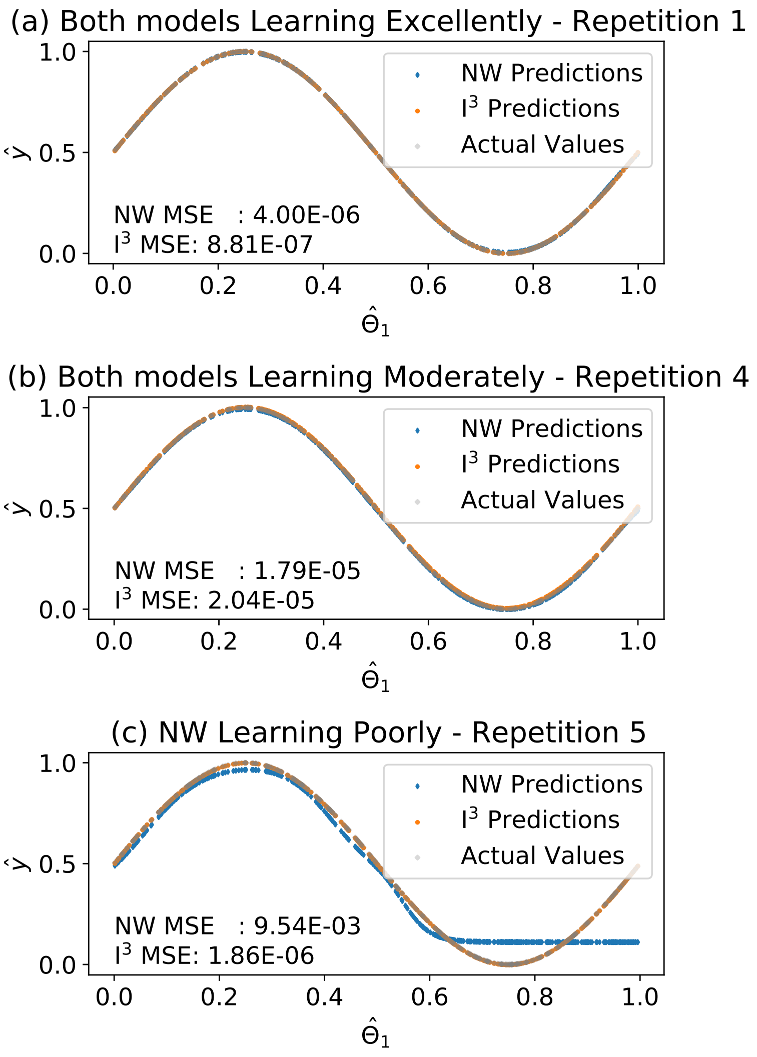


Figure 5.1: Scatter plot of prototypical example of learning a function (a) Excellent learning (b) Moderate learning (c) NW Learning poorly

Table 5.2: Comparison of models with two different initializations (Kanneganti et al., 2021). For Noise level, N, L, M, and H correspond to no, low, medium, and high noise, respectively. M-W $p - value_{NW}$ is the p-value of the Mann-Whitney U test for NW and I^3.

Input	Output	Noise Level	M-W $p - value_{NW}$	Mean MSE		Median MSE	
				NW	I^3	NW	I^3
$\hat{\theta}_1$	$\hat{x}$	N	*3.019e-11*	5.042e-06	5.085e-05	4.330e-06	4.299e-05
		L	*0.002*	1.021e-03	1.141e-03	1.000e-03	1.091e-03
		M	0.185	3.919e-03	4.074e-03	3.901e-03	4.036e-03
		H	0.245	8.579e-03	8.850e-03	8.579e-03	8.618e-03
	$\hat{y}$	N	*1.336e-05*	2.018e-03	4.604e-06	6.180e-06	1.849e-06
		L	0.108	2.973e-03	1.053e-03	1.107e-03	1.058e-03
		M	0.245	5.255e-03	4.141e-03	4.243e-03	4.130e-03
		H	0.641	9.827e-03	9.002e-03	9.030e-03	8.954e-03
$\hat{\theta}_2$	$\hat{x}$	N	*5.572e-10*	1.891e-05	6.866e-05	9.755e-07	5.219e-05
		L	*2.278e-05*	1.091e-03	1.338e-03	1.063e-03	1.232e-03
		M	*6.097e-03*	4.080e-03	4.395e-03	4.002e-03	4.362e-03
		H	*0.030*	8.827e-03	9.183e-03	8.701e-03	9.083e-03
	$\hat{z}$	N	*0.009*	1.382e-05	8.440e-06	1.500e-06	3.016e-06
		L	0.162	1.028e-03	1.081e-03	1.020e-03	1.070e-03
		M	0.620	3.914e-03	3.979e-03	3.869e-03	3.887e-03
		H	0.491	8.510e-03	8.594e-03	8.265e-03	8.609e-03
$\hat{\theta}_3$	$\hat{x}$	N	*3.988e-04*	8.171e-04	5.566e-06	5.789e-07	2.887e-05
		L	0.853	1.869e-03	1.069e-03	1.052e-03	1.072e-03
		M	0.994	4.862e-03	4.114e-03	4.062e-03	4.098e-03
		H	0.717	9.513e-03	8.728e-03	8.619e-03	8.790e-03
$\hat{\theta}_1, \hat{\theta}_2$	$\hat{x}$	N	*1.247e-04*	2.058e-04	2.947e-04	1.004e-04	2.672e-04
		L	*1.370e-03*	1.346e-03	1.537e-03	1.290e-03	1.481e-03
		M	*0.004*	4.438e-03	4.748e-03	4.413e-03	4.693e-03
		H	0.864	9.482e-03	9.484e-03	9.302e-03	9.267e-03
	$\hat{y}$	N	*2.033e-09*	2.192e-04	1.007e-03	1.219e-04	8.345e-04
		L	*8.890e-10*	1.352e-03	2.475e-03	1.294e-03	2.205e-03
		M	*5.091e-06*	4.387e-03	5.449e-03	4.353e-03	5.057e-03
		H	*3.324e-06*	8.922e-03	1.051e-02	8.868e-03	9.989e-03

Chapter 6

Discussion

This chapter discusses the interpretation of results in Chapter 5 linking the research questions and hypothesis in Chapter 3. This chapter has four sections. First section discusses about structural issues, second section discusses initialization issues, third chapter discusses the effect of noise and the final section discusses other interpretations that are observed.

6.1 Structural issues

6.1.1 Structural transferability

The prototype corresponding to *cosine* is used to approximate Equation 2.3. Similarly, the prototype corresponding to *sine* is used to approximate Equation 2.4, Equation 2.7. The best error reported (Table 5.1) to approximate Equation 2.3, Equation 2.4, Equation 2.7 is $7.964e-07$, $2.052e-07$, $2.728e-07$ respectively.

Response: Above errors indicate that the structure recommended by I^3 can be used to approximate above equations in robot kinematics.

6.1.2 Extensibility to more hidden layers

Two hidden layers are used to approximate Equation 2.5, Equation 2.6, Equation 2.8, Equation 2.9, Equation 2.10, Equation 2.11, and Equation 2.12. The best errors reported for these equations are $1.232e-07$, $1.012e-07$, $2.650e-05$, $3.558e-05$, $3.381e-06$, $3.211e-06$, and $3.459e-06$ respectively. Three hidden layers are used to approximate Equation 2.13, and the best error reported is $4.668e-06$. Four hidden layers are used to approximate Equation 2.14, and the best error reported is $6.619e-05$.

Response: Above errors are very low even for more complicated equations involving more hidden layers and indicates that the I^3 is sufficient to allow for multilayer feedforward ANNs with more hidden layers. This extensibility of I^3 allows to apply this approach for this new domain.

6.2 Initialization issues

6.2.1 Domain transferability

For domain transferability we would like to compare the performance of initial values recommended by the two prototype functions. The mean error for Equation 2.3 using NW, Xavier, and I^3 initializations is 5.042e-06, 1.779e-05, 5.085e-05 respectively. Similarly, the mean error for Equation 2.4 using NW, Xavier, and I^3 initializations is 2.018e-03, 7.290e-06, 4.604-06 respectively. For both these equations, we could observe that I^3 initialization performed well w.r.t. other state-of-the-art approaches.

While approximating most of the equations, we observed that the initial values

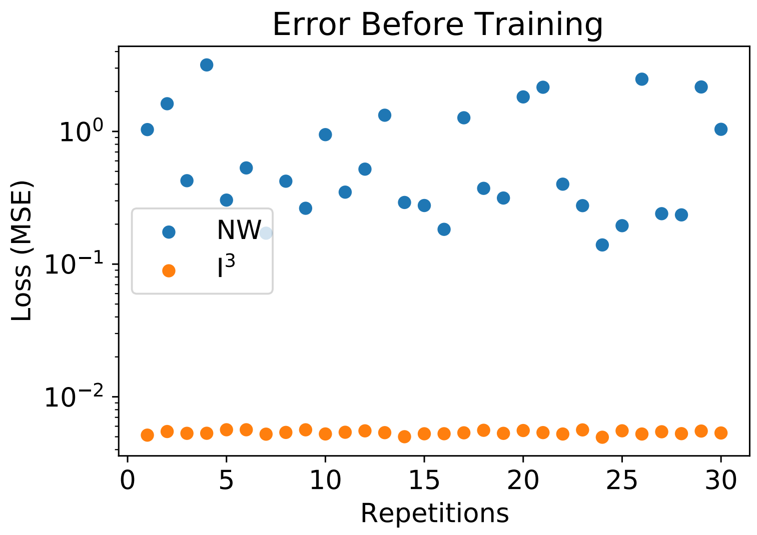


Figure 6.1: Initial errors reported for $\hat{\theta}_1$ as input and $\hat{x}$ as output.

recommended by the I^3 results in lower initial error (error before the start of training process) as seen in Fig. 6.1, Fig. 6.2. Having lower initial error helps to converge quickly to optimal solution; However, it is not always guaranteed to converge quickly to optimal solution. Approximating $\hat{x}$ given $\hat{\theta}_1$ (initial errors in Fig. 6.1) requires one hidden layer. Approximating $\hat{y}$ given $\hat{\theta}_1$,$\hat{\theta}_3$ (initial errors in Fig. 6.2) requires three hidden layer. I^3 initialization continues to have lower initial error even with increase in number of hidden layers.

Response: Initial values recommended by the I^3 approach results in lower initial error.

6.2.2 Concatenating prototypes

For I^3, we observed that the models with one and two hidden layers are *stable*, i.e., ANN model weights and biases do not have extremely large updates during training. However, models with higher hidden layers, Equation 2.13 (three hidden layers) and Equation 2.14 (four hidden layers) are unstable for some epochs of

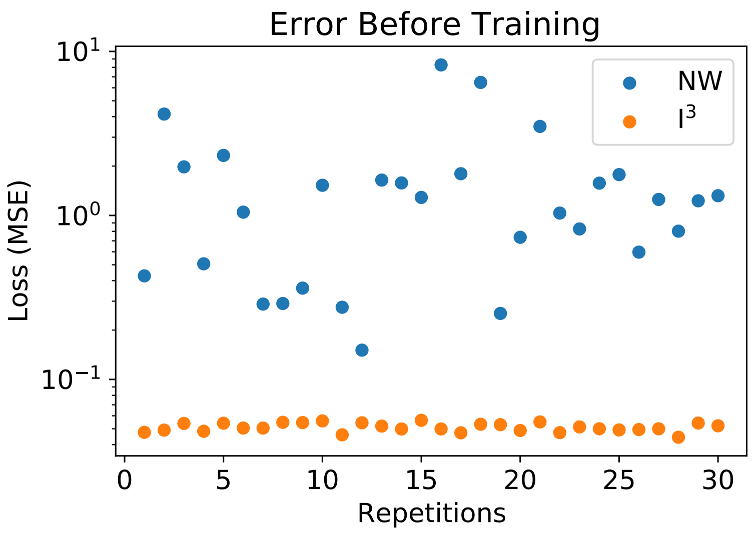


Figure 6.2: Initial errors reported for $\hat{\theta}_1$,$\hat{\theta}_3$ as input and $\hat{y}$ as output.

training. This can be observed from Fig. 6.3 and Fig. 6.4. I am inferring that this unstability in models is due to large initial values in I^3 which leads to exploding gradients.

While approximating Equation 2.13, we observed that only few repetitions have unstable learning and we also observed that the unstability in ANN occurred only once. However, while approximating Equation 2.14, we observed that many repetitions have unstable learning and we also observed that the unstability in these ANNs occurred more than once and can be observed from Fig. 6.4. This also infers that models are likely to become unstable with increase in number of hidden layers.

Response: We infer that weights could explode and lead to exploding gradients with larger hidden layers as a result of condensing layers and concatenating prototypes.

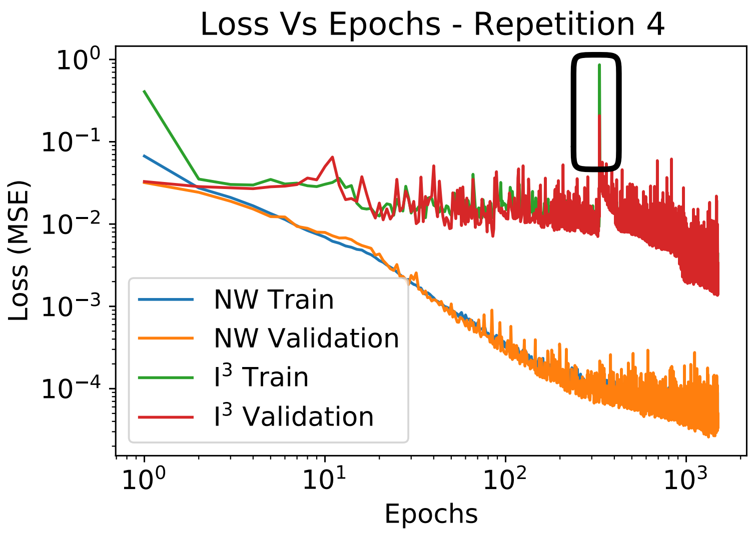


Figure 6.3: History of error Vs epochs for Equation 2.13. The spike in the black rectangular box indicates the large updates in ANN weights and biases during training.

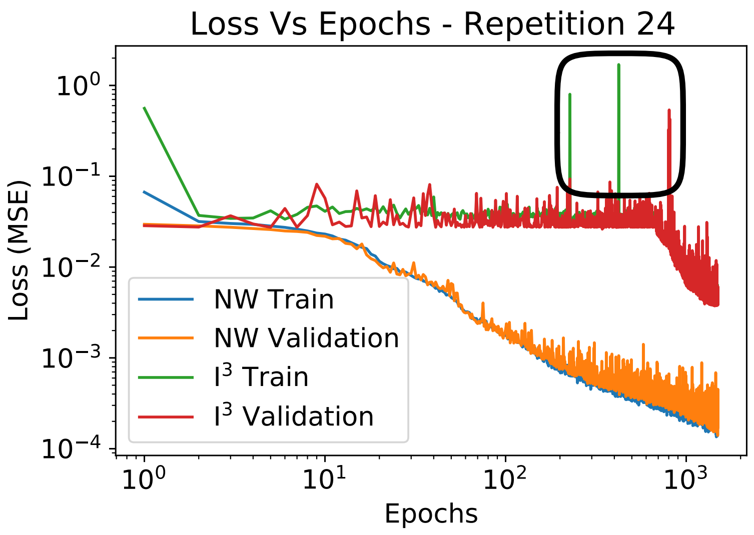


Figure 6.4: History of error Vs epochs for Equation 2.14. The spike in the black rectangular box indicates the large updates in ANN weights and biases during training.

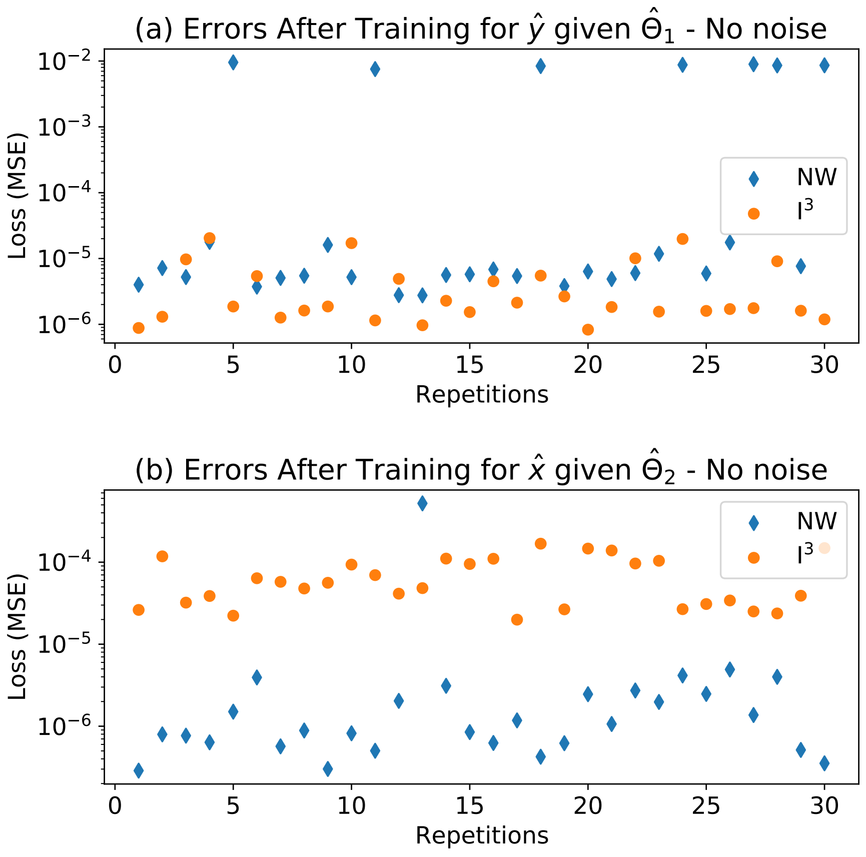


Figure 6.5: Errors reported while training across all repetitions (a) $\hat{y}$ given $\hat{\theta}_1$ (b) $\hat{x}$ given $\hat{\theta}_2$.

6.2.3 Performance evaluation

This subsection discusses performance w.r.t MSE and also being consistent, i.e., does not get stuck at local optima.

For single input equations, w.r.t MSE all the three initializations performed well as each of those initializations performed well for one equation or the other. However, coming to higher inputs (two and three), we can observe that NW and Xavier initializations have lower errors than I^3 and these results are statistically significant.

While approximating some functions, Table 5.1 strongly indicates that I^3 ini-

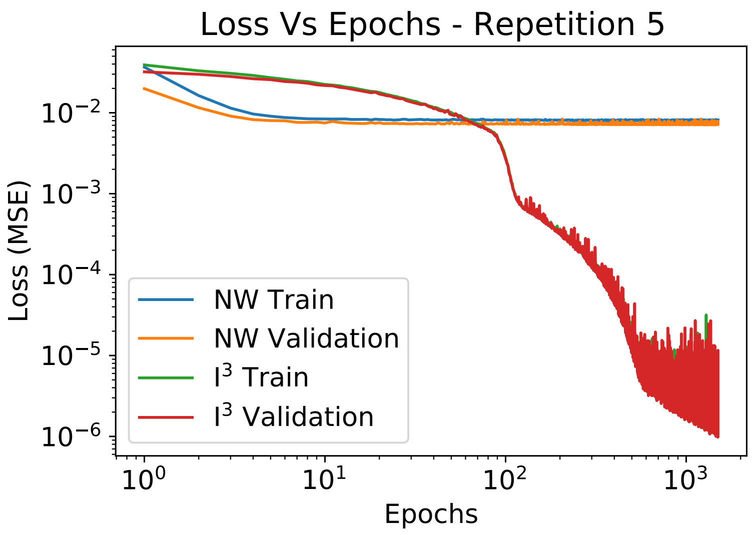


Figure 6.6: History of errors when NW stuck at local optima while training $\hat{y}$ given $\hat{\theta}_1$.

tialization performs consistently and converge to reasonable accuracy for all the repetitions. However, from Table 5.1, Fig. 6.5(a), and Fig. 6.5(b) we can observe that NW initialization results in model stuck at local optima for one or more repetitions. This is also observed when NW outperforms I^3 initialization in many repetitions, this can be observed for 13th repetition in Fig. 6.5(b). Fig. 6.6 shows the progression of error during the training, it is a prototypical example of NW being stuck at local optima. NW does not have just one initial values but rather has multiple initial values. We observed that one or more of these initial values are resulting the model to stuck at local optima. Besides being informative, interactive, and interpretable, I^3 models also help minimize the error; it is suggested to model using both approaches as I^3 helps validate if the NW model got stuck at local optima.

Response: I^3 initialization performs consistently and converge to reasonable error for all the repetitions (when trained for 1500 epochs).

6.3 Effect of noise

6.3.1 Robustness to noise

From Table 5.2, we can observe that noise does not favor any initialization method. Noise does increase the amount of error and it reduce the likelihood that either method will statistically significantly outperform the other. Both NW and I^3 approach seem to be equally affected by noise levels. There wasn't a strong distinguishing factor between the algorithms based on noise.

Response: Noise does not favor any initialization method.

Chapter 7

Conclusions

This work demonstrates robot kinematics as an application to I^3. Below are some of the conclusions drawn from this work.

In this work we have shown that the structure recommended by Pei et al. is capable of approximating equations in other domain (robot kinematics). Moreover, in this work we have extended Pei et al. to three and four hidden layers and also to three inputs. Results also indicate that the structure recommended by Pei et al. is capable of extending to more hidden layers. For single input equations, all the three initializations performed equivalently well and have low error after training. However, models trained using higher inputs (two and three) tend to have lower trained error for Xavier and NW initializations. We also observed that some of the models trained using three, four hidden layers are unstable during training (when trained for 1500 epochs using Adam optimizer). On the other hand, Pei et al. helps to be consistent and prevents from getting stuck at local optima.

Being able to adopt to a new domain (robot kinematics) helps to advance the applicability of Pei et al. to other domains. In this work we have demonstrated the Pei et al. for higher number (three) of inputs, this allows to model multi-dimensional input spaces. Being able to extend to higher number (three and four) of hidden layers allows to model more complex equations. Models developed in this work used Adam optimizer and these models learned well to approximate the equations and this facilitates the way for training deeper ANNs.

We also observed that both NW and Pei et al. seem to be equally affected by noise. In this work we have demonstrated that limited number (four) of prototypes can approximate twelve equations. In fact, these four prototypes can approximate even more equations formed by the combination of these prototypes. In this work, we have also extended the Pei et al. to work for multiple cycles.

We see the power of informed, interactive, and interpretable ML and developed the terminology of I^3 that tries to mean in accordance with the ideas that are put together. Bringing together the ideas of informed, interactive, and interpretable ML and developing the terminology of I^3 makes this work more accessible for broader audience.

We saw the potential of the Pei et al. approach for a broader application, and took the risk for the little-known work to Computer Science ML community. This study not only explores a new domain beyond where the Pei et al. approach had been applied, but also provides some much needed new interpretation and presentations of terminologies and even way of thinking in the original work to facilitate broader applications and better acceptance especially in Computer Science ML community.

Chapter 8

Future Work

This chapter has two sections. First section deals with theoretical advancements and the second section deals with practical advancements.

8.1 Theoretical advancements

This section discusses some of the future work that could help to develop the theory of Pei et al.

Deeper ANNs (three and four-layered-ANNs) trained using Pei et al. have unstable learning for some portion of training. In future we would like to observe weights, biases of ANNs when they are unstable (the portion of training when weights change drastically) and this provides more information about the reasons that is leading to unstable learning. In this work, there could be multiple possibilities for weights and biases to cause exploding gradients. Below effects can be examined further:

1. Explore the effect of concatenating prototypes on weights and biases.

2. Explore the effect of condensing layers. This can be studied by comparing the performance (from history of error Vs epoch figures) of both condensed and non-condensed ANNs.

3. Explore if any specific prototype(s) leads to unstable learning.

8.2 Practical advancements

Below are some of the future work that extends practical advancements to this works.

In this work, we have modeled the forward kinematic equations of simplified PUMA robotic arm, i.e, we have used three input angles (joints) - θ_1, θ_2, θ_3 and upto four hidden layers. A typical PUMA robotic arm can be equipped with a spherical wrist and has six input angles, the other three input angles (θ_4, θ_5, θ_6) are used to determine roll, pitch, and yaw. By modeling forward kinematics of complete robotic arm helps to determine roll, pitch, yaw and more importantly extends the work to more inputs and more hidden layers.

In this work, we have modeled equations for forward kinematics of robotic arm. In robot kinematics, there is also *Inverse kinematics* which calculates the joint variables given the desired location of the end effector. One of the challenges of inverse kinematics is, unlike forward kinematics there could be multiple solutions (combination of joint angles) for a given input (location of end effector). However, using Pei et al. for inverse kinematics makes this more meaningful and interpretable.

In this work, number of inputs have been gradually increased from one to three. However, all these models predict a single output value ($\hat{x}$ or $\hat{y}$ or $\hat{z}$). In future, we would like to design models that have multiple inputs and have multiple outputs. Designing such models decreases the total number of models required to calculate all the outputs.

Noise model used in this work is crude and is not actually a good representation of overall types of errors we would likely have for a real robotic arm. Using real world data will help us to have more interpretation of functioning of robot arms in real world applications.

In this work, we have demonstrated one capable model (architecture and initial values) for each target function. We did not have time to explore optimal designs. In the future, we will experiment and analyze the performance of other capable Pei et al. models.

Bibliography

Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, 2019.

G. Cybenko. Approximation by superpositions of sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

Howard B. Demuth, Mark H. Beale, Orlando De Jess, and Martin T. Hagan. *Neural Network Design.* Martin Hagan, Stillwater, OK, USA, 2nd edition, 2014. ISBN 0971732116.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Dario Floreano and Claudio Mattiussi. *Bio-inspired artificial intelligence: theories, methods, and technologies.* MIT press, 2008.

Henri P Gavin. The levenberg-marquardt algorithm for nonlinear least squares curve-fitting problems. *Department of Civil and Environmental Engineering, Duke University*, 19, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang. XAI - explainable artificial intelligence. *Science Robotics*, 4(37):eaay7120, 2019.

Kris Hauser. Robotic systems, 2020. URL `https://motion.cs.illinois.edu/RoboticSystems/`.

K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.

Dean Frederick Hougen, Jin-Song Pei, and Sai Teja Kanneganti. Toward interpretable machine learning for understanding epidemic data. In *2020 IEEE*

International Conference on Big Data; IEEE International Workshop on Fair and Interpretable Learning Algorithms, pages 3677–3681, December 2020. doi: 10.1109/BigData50022.2020.9377834.

Liu Jiang, Shixia Liu, and Changjian Chen. Recent research advances on interactive machine learning. *Journal of Visualization*, 22(2):401–417, 2019.

Lee K Jones. The computational intractability of training sigmoidal neural networks. *IEEE Transactions on Information Theory*, 43(1):167–173, 1997.

Sai Teja Kanneganti, Jin-Song Pei, and Dean Frederick Hougen. Developing interpretable machine learning for forward kinematics of robotic arms. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–09, 2021. doi: 10.1109/SSCI50451.2021.9660074.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Rudolf Kruse, Christian Borgelt, Christian Braune, Sanaz Mostaghim, Matthias Steinbrecher, Frank Klawonn, and Christian Moewes. *Computational intelligence*. Springer, 2011.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3): 31–57, 2018.

J. Makhoul, A. El-Jaroudi, and R. Schwartz. Formation of disconnected decision regions with a single hidden layer. In *International Joint Conference on Neural Networks*, volume I, pages 455–460, June 1989.

Patrick E. McKnight and Julius Najab. Mann-Whitney U test. In *The Corsini Encyclopedia of Psychology*. American Cancer Society, 2010. ISBN 9780470479216. doi: https://doi.org/10.1002/9780470479216.corpsy0524.

Bernhard Mehlig. *Machine Learning with Neural Networks*. Cambridge University Press, Oct 2021. ISBN 9781108494939. doi: 10.1017/9781108860604. URL `http://dx.doi.org/10.1017/9781108860604`.

Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. Interpretable machine learning–a brief history, state-of-the-art and challenges. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 417–431. Springer, 2020.

Jorge J Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, 2019.

Derrick Nguyen and Bernard Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *1990 International Joint Conference on Neural Networks*, pages 21–26. IEEE, 1990.

J. S. Pei. *Parametric and Nonparametric Identification of Nonlinear Systems*. Ph.d. dissertation, Columbia University, 2001.

J. S. Pei and E. C. Mai. Constructing multilayer feedforward neural networks to approximate nonlinear functions in engineering mechanics applications. *ASME Journal of Applied Mechanics*, 75, 2008.

J. S. Pei, J. P. Wright, and A. W. Smyth. Mapping polynomial fitting into feedforward neural networks for modeling nonlinear dynamic systems and beyond. *Computer Methods in Applied Mechanics and Engineering*, 194(42-44): 4481–4505, 2005.

J. S. Pei, E. C. Mai, and J. P. Wright. Mapping some functions and four arithmetic operations to multilayer feedforward neural networks. In *SPIE International Symposia Smart Structures & Materials/NDE*, 2008.

J. S. Pei, E. C. Mai, J. P. Wright, and S. F. Masri. Mapping some functions and four arithmetic operations to multilayer feedforward neural networks. *Nonlinear Dynamics*, 71(1-2):371–399, 2013.

J. S. Pei, D. F. Hougen, S. T. Kanneganti, J. P. Wright, E. C. Mai, A. W. Smyth, S. F. Masri, A. Derkevorkian, F. Gay-Balmaz, and L. Komini. Interpretable machine learning for function approximation in structural health monitoring. In A. Cury, D. Riberiro, F. Ubertini, and M. D. Todd, editors, *Structural Health Monitoring Based on Data Science Techniques*. Springer, 2021. in press.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Medicine Intelligence*, 1:206–215, 2019.

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys*, 16:1–85, 2022.

E. D. Sontag. Feedforward nets or interpolation and classification. *Journal of Computer and System Sciences*, 45:20–48, 1992.

Mark K. Transtrum and James P. Sethna. Improvements to the Levenberg-Marquardt algorithm for nonlinear least-squares minimization, 2012.

Simon Vollert, Martin Atzmueller, and Andreas Theissler. Interpretable machine learning: A brief survey from the predictive maintenance perspective. In *2021 26th IEEE international conference on emerging technologies and factory automation (ETFA)*, pages 01–08. IEEE, 2021.

Laura Von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan, Giesselbach, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Julius Pfrommer, Annika Pick, Rajkumar Ramamurthy, Michal Walczak, Jochen Garcke, Christian Bauckhage, and Jannis Schuecker. Informed machine learning - a taxonomy and survey of integrating prior knowledge into learning systems. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 2021.

Yu Zhang, Peter Tiňo, Aleš Leonardis, and Ke Tang. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2021.

Appendix A

Appendix

A.1 Initial parameters for other equations using $\mathbf{I}^3$

This section discusses the architecture and initial values for equations which are not discussed earlier in Section 4.4

c. $\hat{x} = 0.5\cos(2\pi\hat{\theta}_2 + \pi/4) + 0.5$

This equation is similar to equation in Subsection 4.4.1.a, except it has a positive phase shift of $\pi/4$. Approximating this equation involves two steps. First, we approximate $2\pi\hat{\theta}_2 + \pi/4$ and then feed this as an input to neural network with initial values as explained in Subsection 4.4.1.a, i.e., $\hat{x} = 0.5\cos(2\pi\hat{\theta}_1) + 0.5$. To approximate $2\pi\hat{\theta}_2 + \pi/4$ we need four hidden nodes, two for approximating $2\pi\hat{\theta}_2$ (first order polynomial term) explained in (Pei et al., 2005) and two for approximating $\pi/4$ (constant term). To approximate $2\pi\hat{\theta}_2$, W_1 are [0.1, -0.1], b_1

are [0, 0], W_2 are [40π, -40π], and b_2 equals [0]. To approximate $\pi/4$, W_1 are [1, -1], b_1 are [0, 0], W_2 are [$\pi/4$, $\pi/4$], and $b_2 = [0]$.

Thus to approximate $2\pi\hat{\theta}_2 + \pi/4$ the initial values used are, W_1 are [0.1, -0.1, 1, -1], b_1 are [0, 0, 0, 0], W_2 are [40π, -40π, $\pi/4$, $\pi/4$], and $b_2 = [0]$. Now, we concatenate the neural network with initial values as explained in Subsection 4.4.1.a. Thus, W_3 are [8.4, 8.4, 1, -1], b_3 are [-4.1, -4.3, 0, 0], W_4 are [-20, 20, 1, 1], and $b_4 = [0]$. This takes the shape of neural network architecture in Fig. A.1. This neural network can be condensed to neural network in Fig. A.2, by removing the hidden layer with nodes having linear activation. Hidden layer can be removed by performing matrix multiplication of incoming and outgoing weights of that layer (Pei et al., 2013).

So, the new weights formed is a product of W_2 and W_3. This causes new weights to explode and results in unstable learning. More about this is discussed in Chapter 6. Thus $W_2 \times W_3$ becomes [[336π, 336π, 40π, -40π], [-336π, -336π, -40π, 40π], [2.1π, 2.1π, $\pi/4$, -$\pi/4$], [2.1π, 2.1π, $\pi/4$, -$\pi/4$]]. Here, first sub list ([336π, 336π, 40π, -40π]) corresponds to outgoing weights from first hidden node of first hidden layer to all the four hidden nodes of second hidden layer respectively. Similarly, second sub list ([-336π, -336π, -40π, 40π]) corresponds to outgoing weights from second hidden node of first hidden layer to all the four hidden nodes of second hidden layer respectively. After condensation, bias values to the nodes remain unchanged. The results shown in Table 5.1 are obtained after condensing a hidden layer with linear activation units.

d. $\hat{z} = 0.5\sin(2\pi\hat{\theta}_2 + \pi/4) + 0.5$

This equation is similar to equation in Subsection 4.4.2.b, except it has a positive phase shift of $\pi/4$. Approximating this equation involves two steps. First, we

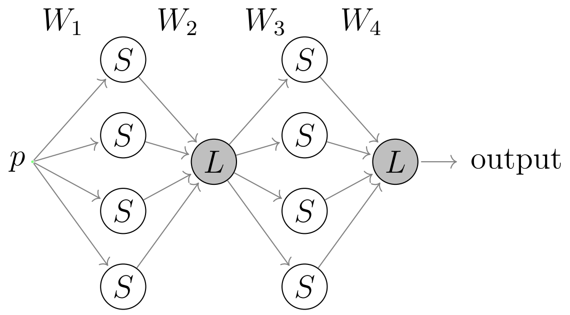


Figure A.1: ANN architecture before condensing hidden layer. S, L nodes correspond to sigmoidal, linear activation functions respectively.

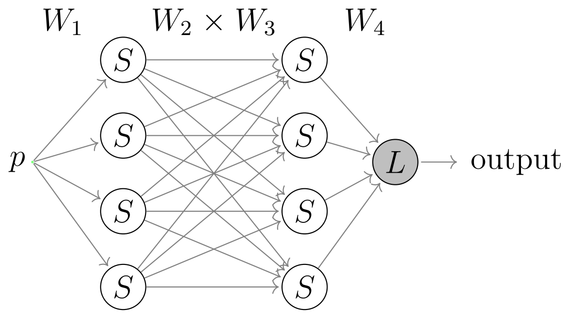


Figure A.2: ANN architecture after condensing hidden layer. S, L nodes correspond to sigmoidal, linear activation functions respectively.

approximate $2\pi\hat{\theta}_2 + \pi/4$ and then feed this as an input to neural network with initial values as explained in Subsection 4.4.2.b. To approximate $2\pi\hat{\theta}_2 + \pi/4$ we need four hidden nodes, two for approximating $2\pi\hat{\theta}_2$ (first order polynomial term) explained in (Pei et al., 2005) and two for approximating $\pi/4$ (constant term). To approximate $2\pi\hat{\theta}_2$, W_1 are [0.1, -0.1], b_1 are [0, 0], W_2 are [40π, -40π], and b_2 equals [0]. To approximate $\pi/4$, W_1 are [1, -1], b_1 are [0, 0], W_2 are [$\pi/4$, $\pi/4$], and $b_2 =$ [0].

Thus to approximate $2\pi\hat{\theta}_2 + \pi/4$ the initial values used are, W_1 are [0.1, -0.1, 1, -1], b_1 are [0, 0, 0, 0], W_2 are [40π, -40π, $\pi/4$, $\pi/4$], and $b_2 =$ [0]. Now, we concatenate the neural network with initial values as explained in Subsection 4.4.2.b. Thus, W_3 are [24, 12, 1, -1], b_3 are [-12, -6, 0, 0], W_4 are [-3.3, 3.3, 0.5, 0.5], and $b_4 =$ [0]. This takes the shape of neural network architecture in Fig. A.1. This neural network can be condensed to neural network in Fig. A.2, by removing the hidden layer with nodes having linear activation. Hidden layer can be removed by performing matrix multiplication of incoming and outgoing weights of that layer (Pei et al., 2013).

So, the new weights formed is a product of W_2 and W_3. This causes new weights to explode and results in unstable learning. More about this is discussed in Chapter 6. Thus $W_2 \times W_3$ becomes [[960π, 480π, 40π, -40π], [-960π, -480π, -40π, 40π], [6π, 3π, $\pi/4$, -$\pi/4$], [6π, 3π, $\pi/4$, -$\pi/4$]]. Here, first sub list ([960π, 480π, 40π, -40π]) corresponds to outgoing weights from first hidden node of first hidden layer to all the four hidden nodes of second hidden layer respectively. Similarly, second sub list ([-960π, -480π, -40π, 40π]) corresponds to outgoing weights from second hidden node of first hidden layer to all the four hidden nodes of second hidden layer respectively. After condensation, bias values to the nodes remain unchanged. The results shown in Table 5.1 are obtained after

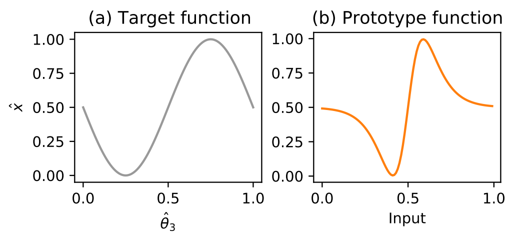


Figure A.3: $\hat{x}$ to be approximated given $\hat{\theta}_3$ (a) Target function to be approximated (b) Prototype function obtained using I^3 to approximate target function.

condensing a hidden layer with linear activation units.

e. $\hat{x} = -0.5\sin(2\pi\hat{\theta}_3) + 0.5$

Fig. A.3(a) shows the function to be approximated. This function has sine term and uses Prototype 2. This function closely takes the shape of function approximated in Section 4.4.2. To approximate this prototype function we require two hidden nodes, similar to Fig. 2.6. The initial parameters to approximate this function follows, W_1 are [20, 10], W_2 are [1, -1]. b_1 takes the values of [0, 0] and b_2 equals to [0]. These initial values can be found in Table. 1 of (Pei and Mai, 2008).

Initial weights to approximate are obtained in two steps. First we scale the prototype function close to our required input and output range. Our required input and output range is from 0 to 1. The input and output range for the prototype is from -1 to 1 and -0.15 to 0.15 respectively. The prototype function can be made closer to Fig. A.3(a), by only considering the input range between [-0.6 to 0.6]; the weights and bias of scaled function can be obtained by substituting values of $\hat{\theta}_1 = 0$ for p = -0.6 and $\hat{\theta}_1 = 0.5$ for $p = 0$. Now, W_1 changes to [24, 12] and b_1 changes to [-12, -6]. To scale the output between -0.5 and 0.5 (the difference between minimum and maximum value of function changes from 0.3 to

1.0), W_2 are multiplied by 3.3, thus W_2 changes to [3.3, -3.3]. Now these weights are better used for our input and output ranges.

In second step, we add a constant of 0.5, this changes output range from -0.5 to 0.5 to our required range of 0 to 1. To add a constant we need two hidden nodes. Now W_1 corresponding to these two hidden nodes are [1, -1], b_1 corresponding to these two hidden nodes are [0, 0], and W_2 corresponding to these two hidden nodes are [0.5, 0.5]. Now, we have a total of 4 hidden nodes, 2 from earlier prototype function and 2 from the the addition of a constant. Thus our final initial parameters are: W_1 are [24, 12, 1, -1], b_1 are [-12, -6, 0, 0], W_2 are [3.3, -3.3, 0.5, 0.5], and $b_2 = [0]$. Now these weights are used to approximate function in Fig. A.3(b). This function is very close to the function we are interested in approximating, i.e., Fig. A.3(a). These parameters takes the shape of ANN architecture in Fig. 4.3.

f. $\hat{x} = 0.5[\cos(2\pi\hat{\theta}_1)\cos(2\pi\hat{\theta}_2 + \pi/4) + 1]$

As discussed in Subsection 4.4.1.a, two hidden nodes can be used to approximate $\cos(2\pi\hat{\theta}_1)$ and these two nodes require $\hat{\theta}_1$ as input. Similarly, two hidden nodes can be used to approximate $\cos(2\pi\hat{\theta}_2 + \pi/4)$ and these two nodes require $\hat{\theta}_2$ as input. The later has a phase shift of $\pi/4$, so, $\cos(2\pi\hat{\theta}_2 + \pi/4)$ can still be approximated by using either $\cos(2\pi\hat{\theta}_2)$ or $\sin(2\pi\hat{\theta}_2)$. The same architecture is capable of learning phase shifts in the input. We choose to approximate using sin prototype. Fig. A.4 shows the architecture used to approximate this equation. First two hidden nodes in the first hidden layer approximates $\cos(2\pi\hat{\theta}_1)$ and next two hidden nodes in that layer approximates $\cos(2\pi\hat{\theta}_2 + \pi/4)$. The missing connections between inputs and first hidden layer has an initial value of 0. So, W_1 are [[8.4, 8.4, 0, 0], [0, 0, 24, 12]], $b1$ are [-4.1, -4.3, -12, -6], and W_2 are [[-20, 0], [20, 0], [0, -3.3], [0, 3.3]]. The nonzero weights corresponding to these four hidden nodes are discussed below.

The nonzero weights corresponding to first two hidden nodes are discussed in this paragraph. First we scale the prototype C to our required input and output range. Our required input and output range is from 0 to 1. The input and output range for the prototype is from -1 to 1 and 0 to 0.25 respectively, which can be seen in Fig. 4.1. The prototype function can be made closer to $\cos(2\pi\hat{\theta}_1)$, by only considering the input range between [-0.6 to 0.6], the weights and bias of scaled function can be obtained by substituting values of $\hat{\theta}_1 = 0$ for $p =$ -0.6 and $\hat{\theta}_1 =$ 0.5 for $p = 0$. Now, W_1 changes to [8.4, 8.4] and b_1 changes to [-4.1, -4.3]. To scale the output between 0 and 1, W_2 are multiplied by 4, thus W_2 changes to [20, -20]. Now these weights are better used for our input and output ranges. In second step, we flip the function by changing the sign of the output. Thereby bringing the prototype function more closer to $\cos(2\pi\hat{\theta}_1)$. Sign of the output can be reversed by changing the sign of the final layer weights, thus W_2 becomes [-20, 20].

Similarly, the nonzero weights corresponding to last two hidden nodes are discussed in this paragraph. First we scale the prototype 2 to our required input and output range. Our required input and output range is from 0 to 1. The input and output range for the prototype is from -1 to 1 and -0.15 to 0.15 respectively, which can be seen in Fig. 4.4. The prototype function can be made closer to $\cos(2\pi\hat{\theta}_2 + \pi/4)$ by only considering the input range between [-0.6 to 0.6]; the weights and bias of scaled function can be obtained by substituting values of $\hat{\theta}_1$ = 0 for $p =$ -0.6 and $\hat{\theta}_1 = 0.5$ for $p = 0$. Now, W_1 changes to [24, 12] and b_1 changes to [-12, -6]. To scale the output between -0.5 and 0.5 (the difference between minimum and maximum value of function changes from 0.3 to 1.0), W_2

are multiplied by 3.3, thus W_2 changes to [3.3, -3.3]. Now these weights are better used for our input and output ranges. In second step, we flip the function by changing the sign of the output. Thereby bringing the prototype function more closer to $\cos(2\pi\hat{\theta}_2 + \pi/4)$. Thus W_2 becomes [-3.3, 3.3].

Approximating two first order polynomial terms requires 8 nodes and it's initial values are mentioned in Table 2 of (Pei et al., 2013). So, $\cos(2\pi\hat{\theta}_1) \times \cos(2\pi\hat{\theta}_2 + \pi/4)$ can be approximated using 8 hidden nodes. Two hidden nodes can be used to add a constant of 1. To approximate a constant term, the last two hidden nodes in third hidden layer are connected to first node in second hidden layer. So, W_3 are [[0.1, 0.1], [-0.1, -0.1], [1, 1], [-1, -1], [0.1, -0.1], [-0.1, 0.1], [1, -1], [-1, 1], [1, 0], [-1, 0]], b_3 are [-10, -10, 0, 0, -10, -10, 0, 0, 0, 0]. W_4 are [5.507e5, 5.507e5, -50.006, -50.006, -5.507e5, -5.507e5, 50.006, 50.006, 1, 1]. W_4 are the final weights, above W_4 weights needs to be further divided by 2 as the whole equation is divided by 2. So, final W_4 are [2.7535e5, 2.7535e5, -25.003, -25.003, -2.7535e5, -2.7535e5, 25.003, 25.003, 0.5, 0.5] and $b_4 = [0]$. This weights and architecture in Fig. A.4 can be condensed to Fig. A.5 by removing the hidden layer with nodes having linear activation function (Pei et al., 2013).

g. $\hat{y} = 0.5[\sin(2\pi\hat{\theta}_1)\cos(2\pi\hat{\theta}_2 + \pi/4) + 1]$

As discussed in Subsection 4.4.1.b, two hidden nodes can be used to approximate $\sin(2\pi\hat{\theta}_1)$ and these two nodes require $\hat{\theta}_1$ as input. Similarly, two hidden nodes can be used to approximate $\cos(2\pi\hat{\theta}_2 + \pi/4)$ and these two nodes require $\hat{\theta}_2$ as input. The later has a phase shift of $\pi/4$, so, $\cos(2\pi\hat{\theta}_2 + \pi/4)$ can still be approximated by using either $\cos(2\pi\hat{\theta}_2)$ or $\sin(2\pi\hat{\theta}_2)$. The same architecture is capable of learning phase shifts in the input. We choose to approximate using sin prototype (prototype 2). Fig. A.4 shows the architecture used to approximate

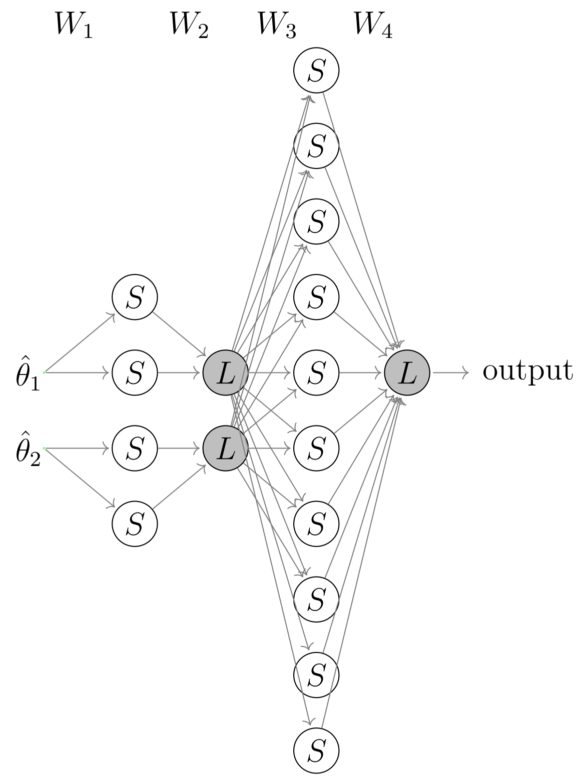


Figure A.4: ANN architecture before condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_2$ as inputs. S, L nodes correspond to sigmoidal, linear activation functions respectively.

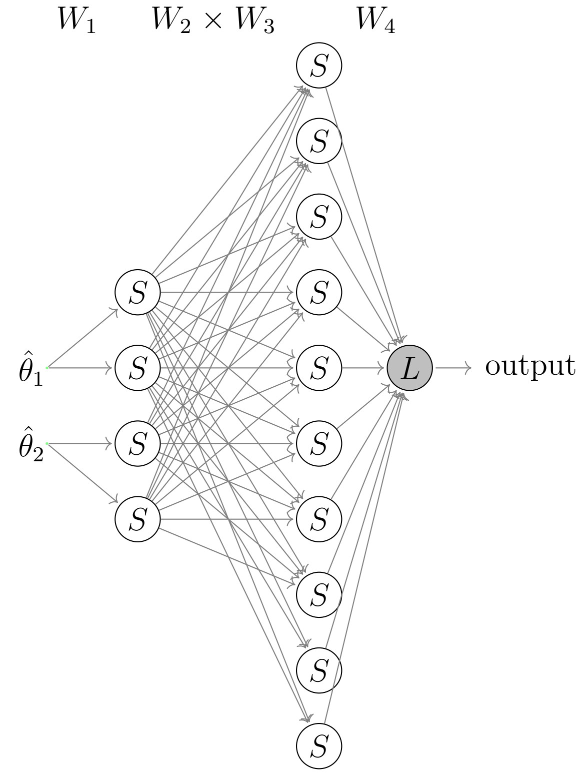


Figure A.5: ANN architecture after condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_2$ as inputs. S, L nodes correspond to sigmoidal, linear activation functions respectively.

this equation. First two hidden nodes in the first hidden layer approximates $\sin(2\pi\hat{\theta}_1)$ and next two hidden nodes in that layer approximates $\cos(2\pi\hat{\theta}_2+\pi/4)$. The missing connections between inputs and first hidden layer has an initial value of 0. So, W_1 are [[24, 12, 0, 0], [0, 0, 24, 12]], $b1$ are [-12, -6, -12, -6], and W_2 are [[-3.3, 0], [3.3, 0], [0, -3.3], [0, 3.3]]. The nonzero weights corresponding to these four hidden nodes are discussed below.

The nonzero weights corresponding to first two hidden nodes and next two hidden nodes are same, as both of them use prototype 2. These weights and biases for two hidden nodes are discussed in this paragraph. First we scale the prototype C to our required input and output range. Our required input and output range is from 0 to 1. The input and output range for the prototype is from -1 to 1 and -0.15 to 0.15 respectively, which can be seen in Fig. 4.4. The prototype function can be made closer to $\sin(2\pi\hat{\theta}_2)$ by only considering the input range between [-0.6 to 0.6]; the weights and bias of scaled function can be obtained by substituting values of $\hat{\theta}_1 = 0$ for $p =$ -0.6 and $\hat{\theta}_1 = 0.5$ for $p = 0$. Now, W_1 changes to [24, 12] and b_1 changes to [-12, -6]. To scale the output between -0.5 and 0.5 (the difference between minimum and maximum value of function changes from 0.3 to 1.0), W_2 are multiplied by 3.3, thus W_2 changes to [3.3, -3.3]. Now these weights are better used for our input and output ranges. In second step, we flip the function by changing the sign of the output. Thereby bringing the prototype function more closer to $\sin(2\pi\hat{\theta}_2)$. Thus W_2 becomes [-3.3, 3.3].

Approximating two first order polynomial terms requires 8 nodes and it's initial values are mentioned in Table 2 of (Pei et al., 2013). So, $\sin(2\pi\hat{\theta}_1) \times \cos(2\pi\hat{\theta}_2 + \pi/4)$ can be approximated using 8 hidden nodes. Two hidden nodes can be used to add a constant of 1. To approximate a constant term, the last two hidden nodes in third hidden layer are connected to first node in second hidden layer. So, W_3 are [[0.1, 0.1], [-0.1, -0.1], [1, 1], [-1, -1], [0.1, -0.1], [-0.1, 0.1], [1, -1], [-1, 1], [1, 0], [-1, 0]], b_3 are [-10, -10, 0, 0, -10, -10, 0, 0, 0, 0]. W_4 are [5.507e5, 5.507e5, -50.006, -50.006, -5.507e5, -5.507e5, 50.006, 50.006, 1, 1]. W_4 are the final weights, above W_4 weights needs to be further divided by 2 as the whole equation is divided by 2. So, final W_4 are [2.7535e5, 2.7535e5, -25.003, -25.003, -2.7535e5, -2.7535e5, 25.003, 25.003, 0.5, 0.5] and b_4 = [0]. This weights and architecture in Fig. A.4 can be condensed to Fig. A.5 by removing the hidden layer with nodes having linear activation function (Pei et al., 2013).

i. $\hat{z} = 0.25\sin(2\pi\hat{\theta}_2) + 0.25\cos(2\pi(\hat{\theta}_2 + \hat{\theta}_3)) + 0.5$

This equation is similar to equation in 4.4.3. So, we will be using first hidden layer nodes to approximate $\hat{\theta}_2$ and $(\hat{\theta}_2 + \hat{\theta}_3)$. So, W_1 and $\hat{\theta}_2$ are same as in 4.4.3. The main difference from 4.4.3 equation is, here $\hat{\theta}_2$ is used to approximate $0.25\sin(2\pi\hat{\theta}_2)$ and $(\hat{\theta}_2 + \hat{\theta}_3)$ is used to approximate $0.25\cos(2\pi(\hat{\theta}_2 + \hat{\theta}_3))$. So, W_1, b_1, W_2, and b_2 are same as 4.4.3. Thus, W_1 are [[0.1, -0.1, 0.1, -0.1, 0, 0], [0, 0, 0, 0, 0.1, -0.1]], $b1$ are [0, 0, 0, 0, 0, 0], and W_2 are [[20, 0], [-20, 0], [0, 20], [0, -20], [0, 20], [0, -20]]. The second hidden layer consists of two linear nodes, so $b2$ are [0, 0].

The third hidden layer consists of ten hidden nodes. First two hidden nodes are used to approximate the first component, i.e, $0.25\sin(2\pi\hat{\theta}_2)$ given $\hat{\theta}_2$ as input. The initial weights and bias for this component are obtained similar to Subsection 4.4.2.a. For approximating this sin function, a regular variant (prototype 3, variant c) is used. Thus, incoming weights becomes [24, 12] and bias of hidden neurons becomes [-12, -6], outgoing weights becomes [-1.65, 1.65]. Intermediate six hidden nodes are used to approximate the second component, i.e, $0.25\cos(2\pi(\hat{\theta}_2+\hat{\theta}_3))$ given $(\hat{\theta}_2+\hat{\theta}_3)$ as input. Prototypes are generally designed to take input within the range of 0 to 1. However, $(\hat{\theta}_2 + \hat{\theta}_3)$ has a range of 0 to 2; to facilitate this there is a modification done while designing the architecture. Four hidden nodes are sufficient to approximate $0.25\cos(2\pi(\hat{\theta}_2))$ (two hidden nodes for one cycle of cos and two hidden nodes for shifting), as the input range is from 0 to 1. However, to facilitate for an input range of o to 2 it requires a total of 6 hidden nodes; We require two hidden nodes to approximate one cycle between 0 to 1, two more hidden nodes to approximate another cycle between 1 and 2, and two hidden nodes for shifting. The final two hidden layers are used to approximate a constant of 0.5.

The parameters corresponding to first 2 hidden nodes of $0.25\cos(2\pi(\hat{\theta}_2 + \hat{\theta}_3))$ are obtained similar to 4.4.1. The incoming weights, bias and outgoing weights of these two hidden neurons becomes [8.4, 8.4], [-4.1, -4.3], and [-10, 10] respectively. The parameters of next 2 hidden nodes are used to approximate cos for the second cycle. The incoming weights, bias and outgoing weights of these two hidden neurons becomes [8.4, 8.4], [-12.5, -12.7], and [-10, 10] respectively. The parameters of next 2 hidden nodes are used to shift a cos curve (generated by earlier 4 hidden units) by 0.25. Thus the incoming weights, bias and outgoing weights of these two hidden neurons becomes [1, -1], [0, 0], and [0.25, 0.25] respectively.

Last two hidden nodes in third hidden layer are used to approximate a constant of 0.5, these nodes are connected to hidden node that outputs $(\hat{\theta}_2 + \hat{\theta}_3)$; Now for these two nodes, the incoming weights becomes [1, -1] and bias of hidden neurons changes to [0, 0], outgoing weights becomes[0.5, 0.5].

Thus, W_3 are [[24, 12, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 8.4, 8.4, 8.4, 8.4, 1, -1, 1, -1]], $b3$ are [-12, -6, -4.1, -4.3, -12.5, -12.7, 0, 0, 0, 0], W_4 are [-1.65, 1.65, -10, 10, -10, 10, 0.25, 0.25, 0.5, 0.5], and b_4 equals to [0]. These weights and architecture in

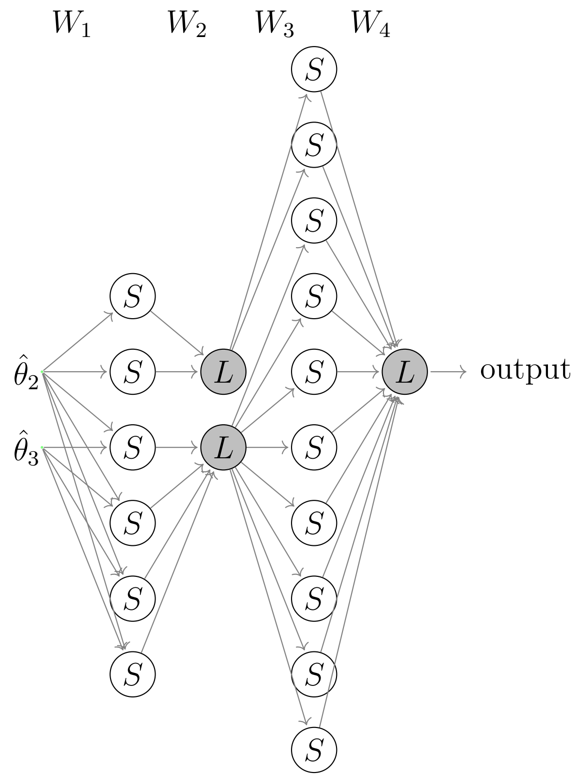


Figure A.6: ANN architecture before condensing hidden layer - for $\hat{\theta}_2, \hat{\theta}_3$ as inputs and $\hat{z}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively.

Fig. A.6 can be condensed to Fig. A.7 by removing the hidden layer with nodes having linear activation function.

j. $\hat{x} = 0.25\cos(2\pi\hat{\theta}_1) * [1 - \sin(2\pi\hat{\theta}_3)] + 0.5$

This equation consists of approximating two parts and then multiplying their products. First part approximates $0.25\cos(2\pi\hat{\theta}_1)$ and the second part approximates $[1 - \sin(2\pi\hat{\theta}_3)]$. Approximating first part uses prototype Cand requires $\hat{\theta}_1$ as input and requires four hidden nodes to approximate it, as discussed below:

Equation (4.2) can be approximated using two hidden nodes, as shown in Fig. 2.6. We choose a small value of Δb equals to 0.1, it is an empirical choice

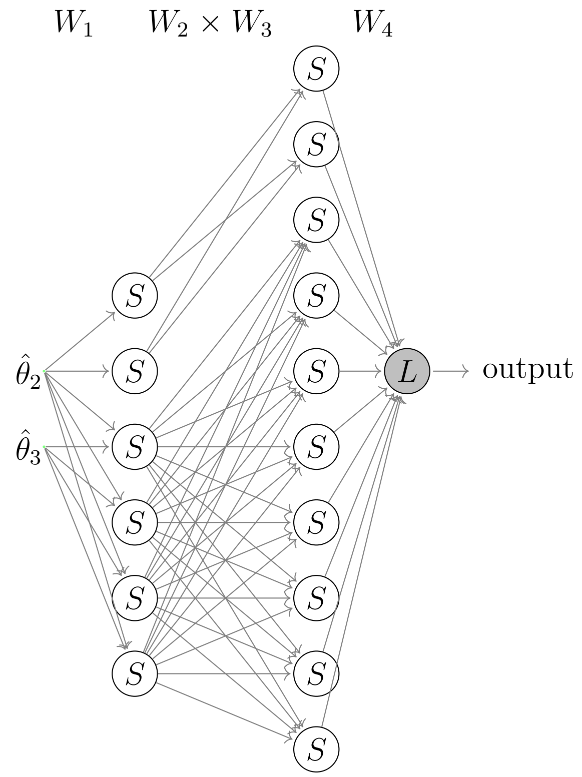


Figure A.7: ANN architecture after condensing hidden layer - for $\hat{\theta}_2, \hat{\theta}_3$ as inputs and $\hat{z}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively.

and needs to be a small value. Based on the above information, W_1 are [7, 7]. From (4.2), we can observe that W_2 are $[\frac{1}{2\Delta b}, -\frac{1}{2\Delta b}]$ respectively. Thus, W_2 becomes [5, -5]. b_1 takes the values of $[b + \Delta b, b - \Delta b]$, b_1 equals [0.1, -0,1], and b_2 equals [0].

Initial weights to approximate are obtained in three steps. First we scale the prototype function to our required input and output range. Our required input and output range is from 0 to 1 and can be observed in Fig. 4.2(a). The input and output range for the prototype is from -1 to 1 and 0 to 0.25 respectively, which can be seen in Fig. 4.1. The prototype function can be made closer to Fig. 4.2(a), by only considering the input range between [-0.6 to 0.6], the weights and bias of scaled function can be obtained by substituting values of $\hat{\theta}_1 = 0$ for $p =$ -0.6 and $\hat{\theta}_1 = 0.5$ for $p = 0$. Now, W_1 changes to [8.4, 8.4] and b_1 changes to [-4.1, -4.3]. To scale the output within the range of 0.5 (-0.25 to 0.25), W_2 are multiplied by 2, thus W_2 changes to [10, -10]. Now these weights are better used for our input and output ranges.

In second step, we flip the function by changing the sign of the output. Thereby bringing the prototype function more closer to equation $0.25\cos(2\pi\hat{\theta}_1)$. Sign of the output can be reversed by changing the sign of the final layer weights, thus W_2 becomes [-10, 10].

In final step, we add a constant of 0.25, to bring this function more closer to the equation $0.25\cos(2\pi\hat{\theta}_1)$. To add a constant we need two hidden nodes as described in Chapter 2, Section 2.3. Now W_1 corresponding to these two hidden nodes are [1, -1], b_1 corresponding to these two hidden nodes are [0, 0], and W_2 corresponding to these two hidden nodes are [0.25, 0.25]. Now, we have a total of 4 hidden nodes, 2 from earlier prototype function and 2 from the the addition of constant. Thus our final initial parameters are: W_1 are [8.4, 8.4, 1, -1], b_1 are

[-4.1, -4.3, 0, 0], W_2 are [-10, 10, 0.25, 0.25], and b_2 equals [0]. Now these weights are used to approximate sub equation $0.25\cos(2\pi\hat{\theta}_1)$.

Approximating second part requires, four hidden nodes. Two of these hidden nodes are used to approximate a constant 1, to add a constant (1) we need two hidden nodes as described in Chapter 2, Section 2.3. Now W_1 corresponding to these two hidden nodes are [1, -1], b_1 corresponding to these two hidden nodes are [0, 0], and W_2 corresponding to these two hidden nodes are [1, 1]. 2 other nodes are used to approximate $-\sin(2\pi\hat{\theta}_3)$ (similar to approximating $\sin(2\pi\hat{\theta}_2)$ in Section A.1) as discussed below:

The equation $-\sin(2\pi\hat{\theta}_3)$ closely takes the shape of the prototype 3, variant c in Fig. 3 of (Pei and Mai, 2008), which is also depicted in Fig. 4.4. To approximate this prototype function we require two hidden nodes, similar to Fig. 2.6. The initial parameters to approximate this function follows, W_1 are [20, 10], W_2 are [1, -1]. b_1 takes the values of [0, 0] and b_2 equals to [0]. These initial values can be found in Table. 1 of (Pei and Mai, 2008). First we scale the prototype 2 function close to our required input and output range. Our required input range is from 0 to 1 and required output range is 2 (-1 to 1). The input and output range for the prototype is from -1 to 1 and -0.15 to 0.15 respectively. The prototype function can be made closer to $-\sin(2\pi\hat{\theta}_3)$, by only considering the input range between [-0.6 to 0.6]; the weights and bias of scaled function can be obtained by substituting values of $\hat{\theta}_1 = 0$ for p = -0.6 and $\hat{\theta}_1 = 0.5$ for p = 0. Now, W_1 changes to [24, 12] and b_1 changes to [-12, -6]. To scale the output within the range of 2 (-1 to 1) (the difference between minimum and maximum value of function changes from 0.3 to 2.0), W_2 are multiplied by 6.6, thus W_2 changes to [6.6, -6.6]. Now these weights are better used for our input and output ranges.

So, finally W_1 are [[8.4, 8.4, 1, -1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 1, 24, 12]], $b1$ are

[-4.1, -4.3, 0, 0, 0, 0, -12, -6], and W_2 are [[-10, 0], [10, 0], [0.25, 0], [0.25, 0], [0, 1], [0, 1], [0, 6.6], [0, -6.6]]. The second hidden layer now consists of two linear nodes, so $b2$ are [0, 0].

We need 10 hidden nodes in third hidden layer, architecture is observed in Fig. A.8. First, 8 hidden nodes are used to approximate the product of above two approximations. Next 2 hidden nodes are used to approximate a constant of 0.5. Thus, W_3 are [[0.1, -0.1, 1, -1, 0.1, -0.1, 1, -1, 0, 0], [0.1, -0.1, 1, -1, -0.1, 0.1, -1, 1, 1, -1]], $b3$ are [10, 10, 0, 0, 10, 10, 0, 0, 0, 0], and W_4 are [5.5076e5, 5.5076e5, -50.0068, -50.0068, $-5.5076e5$, $-5.5076e5$, 50.0068, 50.0068, 0.5, 0.5], and $b4$ is [0]. These weights and architecture in Fig. A.8 can be condensed to Fig. A.9.

k. $\hat{y} = 0.5\sin(2\pi\hat{\theta}_1) * \cos(2\pi(\hat{\theta}_3 + 1/8)) + 0.5$

This equation consists of approximating two parts and then multiplying their products. First part approximates $0.25\sin(2\pi\hat{\theta}_1)$ and the second part approximates $\cos(2\pi\hat{\theta}_3 + 1/8)$.

Approximating first part requires $\hat{\theta}_1$ as input and requires two hidden layers to approximate it. Two hidden nodes are used in first hidden layer and it is used to approximate just $\hat{\theta}_1$. Two hidden layers are used in second hidden layer to approximate $0.25\sin(2\pi\hat{\theta}_1)$. We require one additional layer in first part than necessary as the target function involves a cosine function of $(2\pi\hat{\theta}_3+1/8)$, number of hidden nodes used for this target function is more than needed as we want to have a fully connected neural network and $(2\pi\hat{\theta}_3 + 1/8)$ controls the depth of the neural network.

Approximating second part requires, two hidden layers with four hidden nodes each. Two hidden units in first hidden layer are used to approximate $\hat{\theta}_1$ and two hidden nodes are used to approximate 1/8. The resultant sum $(\hat{\theta}_3 + 1/8)$ is fed

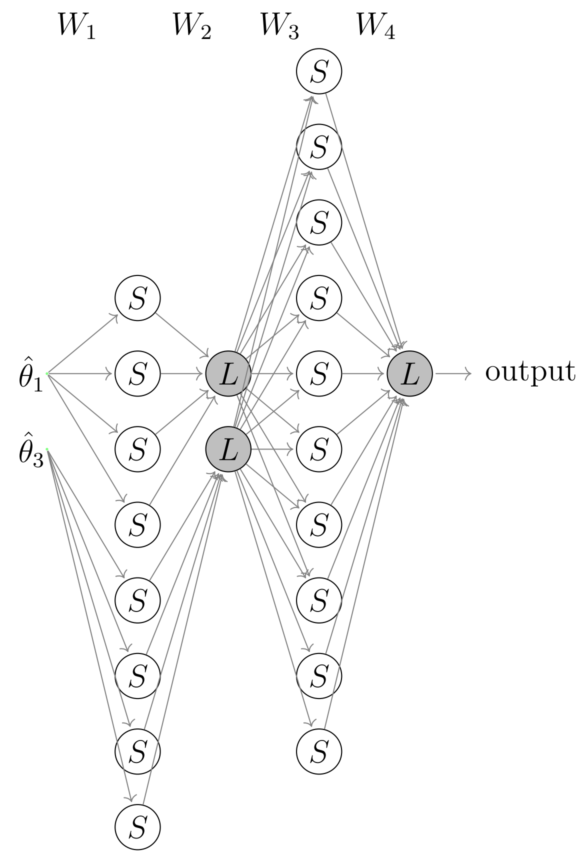


Figure A.8: ANN architecture before condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_3$ as inputs and $\hat{x}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively.

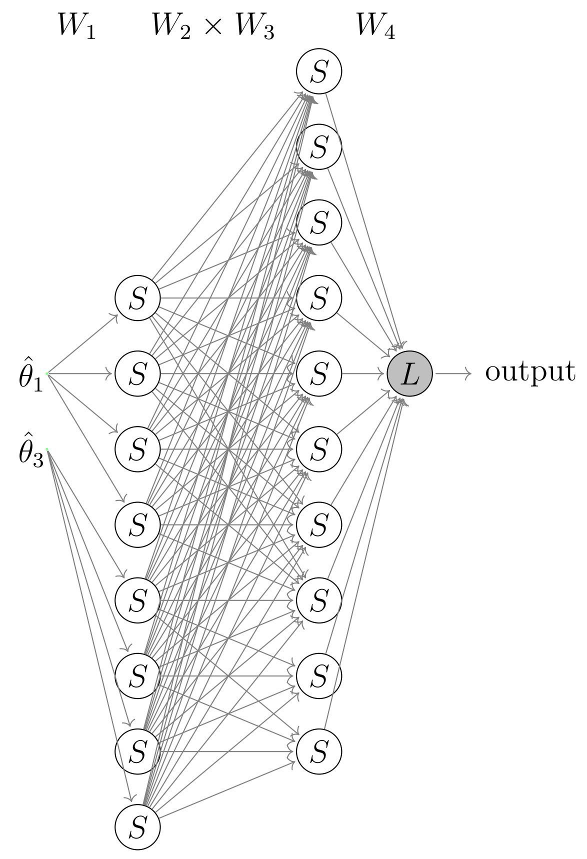


Figure A.9: ANN architecture after condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_3$ as inputs and $\hat{x}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively.

as an input to second hidden layer with four hidden units, these hidden units are used to approximate cos of the input, i.e., $\cos(2\pi\hat{\theta}_3 + 1/8)$.

Two of these hidden nodes are used to approximate a constant 1 and 2 other nodes are used to approximate $-\sin(2\pi\hat{\theta}_3)$ (similar to approximating $\sin(2\pi\hat{\theta}_2)$ in Section A.1). So, W_1 are [[8.4, 8.4, 1, -1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 1, 24, 12]], $b1$ are [-4.1, -4.3, 0, 0, 0, 0, -12, -6], and W_2 are [[-10, 0], [10, 0], [0.25, 0], [0.25, 0], [0, 1], [0, 1], [0, 6.6], [0, -6.6]]. The second hidden layer now consists of two linear nodes, so $b2$ are [0, 0].

We need 10 hidden nodes in third hidden layer, architecture is observed in Fig. A.8. First, 8 hidden nodes are used to approximate the product of above two approximations. Next 2 hidden nodes are used to approximate a constant of 0.5. Thus, W_3 are [[0.1, -0.1, 1, -1, 0.1, -0.1, 1, -1, 0, 0], [0.1, -0.1, 1, -1, -0.1, 0.1, -1, 1, 1, -1]], $b3$ are [10, 10, 0, 0, 10, 10, 0, 0, 0, 0], and W_4 are [5.5076e5, 5.5076e5, 50.0068, 50.0068, 5.5076e5, 5.5076e5, 50.0068, 50.0068, 0.5, 0.5], and $b4$ is [0]. These weights and architecture in Fig. A.10 can be condensed to Fig. A.11.

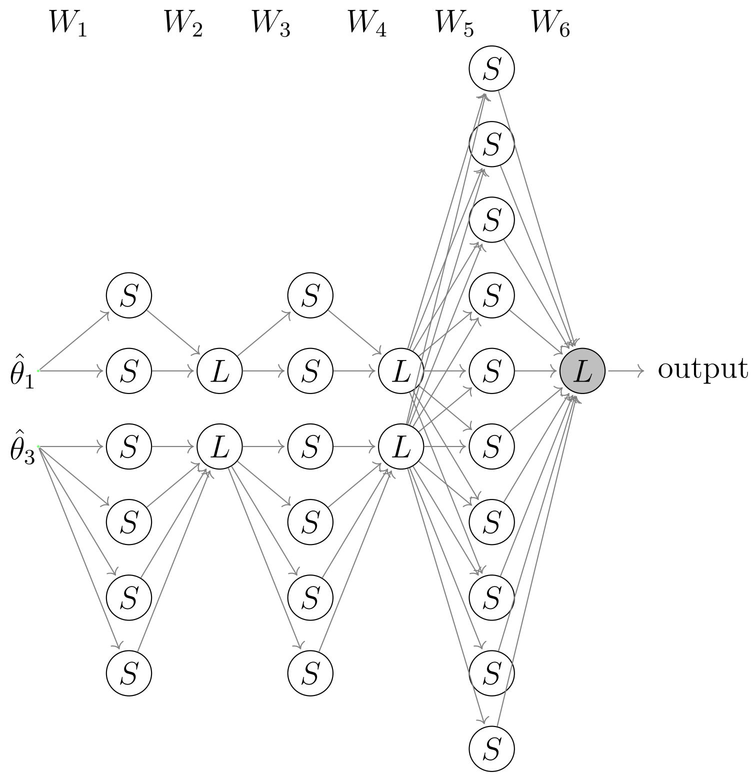


Figure A.10: ANN architecture before condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_3$ as inputs and $\hat{y}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively.

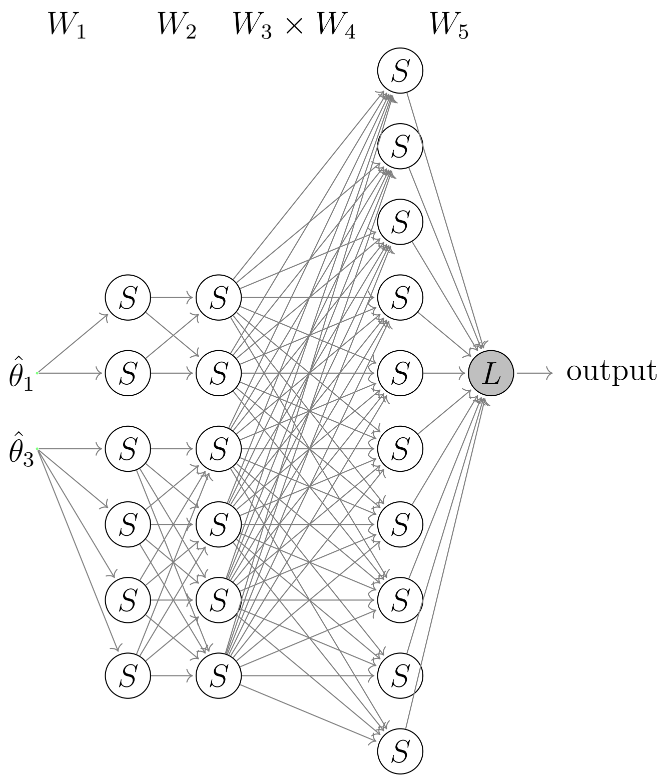


Figure A.11: ANN architecture after condensing hidden layer - for $\hat{\theta}_1, \hat{\theta}_3$ as inputs and $\hat{y}$ as output. S, L nodes correspond to sigmoidal, linear activation functions respectively.