

Machine Learning Framework for Causal Modeling for Process Fault Diagnosis and
Mechanistic Explanation Generation

Abhishek Sivaram

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2023

© 2022

Abhishek Sivaram

All Rights Reserved

Abstract

Machine Learning Framework for Causal Modeling for Process Fault Diagnosis and
Mechanistic Explanation Generation

Abhishek Sivaram

Machine learning models, typically deep learning models, often come at the cost of explainability. To generate explanations of such systems, models need to be rooted in first-principles, at least mechanistically. In this work we look at a gamete of machine learning models based on different levels of process knowledge for process fault diagnosis and generating mechanistic explanations of processes. In chapter 1, we introduce the thesis using a range of problems from causality, explainability, aiming towards the goal of generating mechanistic explanations of process systems. Chapter 2 looks at an approach for generating causal models purely through data-centric approach, with minimal process knowledge with respect to equipment connectivity and identifying causality in the domains. These causal models generated can be utilized for process fault diagnosis.

Chapter 3 and chapter 4 show how deep learning models can be used for both classification for process fault diagnosis and regression. We see that depending on the hyperparameters, i.e., purely the breadth and depth of a neural network, the learned hidden representations vary from a simple set of features, to more complex sets of features. While these hidden representations may be exploited to aid in classification and regression problems, the true explanations of these representations do not correlate with mechanisms in the system of interest. There is thus a requirement to add more mechanistic information

about the features generated to aid in explainability.

Chapter 5 shows how incorporating process knowledge can aid in generating such mechanistic explanations based on automated variable transformations. In this chapter we show how process knowledge can be used to generate features, or model forms to generate explainable models. These models have the ability of extracting the true models of the system from the model knowledge provided.

Table of Contents

Acknowledgments	ix
Chapter 1: Introduction and Background	1
Chapter 2: From data to causal models	5
2.1 Transfer Entropy as a Measure of Causality	8
2.1.1 Generating digraph based on transfer entropy	11
2.2 Tennessee Eastman Benchmark Process	14
2.3 Hierarchical framework for developing causal maps	17
2.3.1 Tier 1: Plant-level DAG	19
2.3.2 Tier 2: Subsystem-level graph with possible cycles	30
2.4 Major Results	33
Chapter 3: Neural Networks for Classification	35
3.1 Mathematical Background	38
3.1.1 Problem Formulation	38
3.1.2 Classification with Neural Networks	40
3.2 Peeking Under the Hood of a Deep Neural Network	42
3.2.1 Feature Extraction: Node-specific Selective Activation of the Input Space	43

3.2.2	Wider vs Deeper Networks: Complexity of Features	45
3.2.3	From Features to Feature Spaces	49
3.2.4	The Final Layer: Separating Hyperplanes for Classification	50
3.2.5	Degeneracy of parameters using softmax activation	52
3.3	How does a Neural Network Learn the Mapping?: From Parts to Whole . . .	56
3.4	Major Results	67
Chapter 4: Neural Networks for Regression		71
4.1	Mathematical Background	73
4.1.1	Problem Formulation	73
4.1.2	Regression with Neural Networks	74
4.2	Peeking Under the Hood of a Neural Network	77
4.2.1	Node-specific Local Approximation	77
4.2.2	Wider vs Deeper Networks: Complexity of Local Approximations . .	80
4.2.3	Degeneracy of Parameters	82
4.3	How does a Neural Network Approximate a Function?: From Parts to Whole	90
4.3.1	Illustrative Example: Sinusoidal Function	90
4.4	Demonstrative Example: Energy Function Landscape	95
4.4.1	Effect of Depth and Width on the Predicted Energy Landscape . . .	98
4.4.2	Node specific local approximations	99
4.5	Major Results	101
Chapter 5: Mechanistic Explanation Generation (MEG)		105
5.1	AI for Mechanistic Explanation Generation – XAI-MEG	105

5.1.1	Temporal and spatio-temporal models for explanation generation . . .	108
5.1.2	Feature extraction from measurements	110
5.1.3	Model Estimator	111
5.1.4	Explanation Generation	112
5.2	Results and Discussion	113
5.2.1	Case Study 1: Simple Harmonic Motion	114
5.2.2	Case Study 2: Damped Simple Harmonic Motion	116
5.2.3	Case Study 3: Lotka-Volterra System	117
5.2.4	Case Study 4: Compartmental Models in Epidemiology – S-I-R model	119
5.2.5	Case Study 5: Convection Systems	120
5.2.6	Case Study 6: Diffusion Systems	122
5.2.7	Case Study 7: Reaction Diffusion Systems	123
5.3	Major Results	125
Chapter 6: Discussion		127
Epilogue		131
References		132

List of Figures

2.1	(a) A simple case study of mixer simulated in MATLAB simulink (b) Significance value matrix and the derived weighted adjacency matrix. Weights greater than threshold are marked in blue. (c) Transfer entropy-based digraph generated for the system	10
2.2	Causal maps (a) and (b) are obtained based on equation (2.8) for examples (c) and (d) in Table 2.1 respectively. It can be seen that many of the significant connections are missing. The reason can be attributed to the presence of feedback loops/recycle streams.	12
2.3	Process flowsheet of the Tennessee Eastman benchmark process, with a decentralized control scheme (As seen in [43])	16
2.4	Dynamic profiles of variables 1, 9, 13 and 20.	18
2.5	(a) Example for a feedback system. All streams except 5 are omitted for analysis. (b) Example for a unit with recycle stream. The subsystem marked by the red dotted line is considered as a single unit and thus, only streams 1 and 3 are included for developing plant-level DAG. (c) Example for the case where 2 units are connected using a recycle stream.	20
2.6	Demerits of data-driven causality detection algorithms. If a data-driven algorithm results in a causal map as shown in (a) for a system S, its true representation could be any of the causal maps shown in (b).	21
2.7	Plant-level causal map constructed based on transfer entropy for the Tennessee Eastman benchmark process (Base case). $w_0 = 0.12$ is used as threshold. . .	22
2.8	Plant-level DAG constructed based on transfer entropy for the Tennessee Eastman benchmark process (Base case) by setting $w_0 = 0.14$ as threshold. . . .	26
2.9	A simple case study to show the direct causality detection algorithm. \hat{W} shows a weight of 1 for the indirect edge, whereas its weight was reduced to 0.34 after transformation.	28
2.10	Final plant-level DAG constructed based on transfer entropy for the Tennessee Eastman benchmark process (base case) after transformation. $w_0 = 0.14$ is used as threshold. The gray arrows are the once which were nullified in comparison to the originally obtained graph	29
2.11	A few examples of subsystem-level causal maps for the Tennessee Eastman case study.	32
2.12	Flowchart of the proposed hierarchical approach.	34
3.1	Activation functions used in this study	42
3.2	Architecture for 2 input neurons and 1 activated neuron	43

3.3	Activation of the network shown in Figure 3.2 for different regions of the input space with different activation functions. The red line shows the set of points satisfying $w_1x_1 + w_2x_2 + b = 0$ ($w_1 = -1, w_2 = 1, b = 0$).	44
3.4	Activation of the network shown in Figure 3.2 for different regions of the input space with different activation functions. The red line shows the set of points satisfying $w_1x_1 + w_2x_2 + b = 0$ ($w_1 = -3, w_2 = 3, b = 0$)	45
3.5	Architecture for 2 input neurons and 1 activated neuron, with 1 hidden layer	46
3.6	Input space activation for each node in the network architecture as shown in Figure 3.5. The values of the weights and biases are chosen to be $w_1, w_2, b = -3, 3, 0, w_3, w_4, c = 3, 3, -3$	47
3.7	Example transformations of the input space to the feature space. The Input space (a) is first rotated and stretched (b), then translated to give the \mathbf{z} vector (c). Note how all the transformations are affine so far. The final nonlinear operation comes from the squishing action based on different activation function on the transformed space to give the final feature space representation (d),(e),(f) and (g). The dark regions in the input space are mapped to dark regions in the subsequent feature space.	50
3.8	Sample architecture for linearly separable problems	51
3.9	(a) Final estimated segregation on the input space. Thickness indicates the magnitude of \mathbf{w}_j for class j . The black lines are constructed using the boundary equation—Equation (3.7). (b) Identified probabilities of each point on the input space	53
3.10	Example datasets for studying the operation of neural networks	58
3.11	Internal internal representation of network with $n = 3, H = 3$ and <i>tanh</i> as the activation function trained on the moons dataset	58
3.12	Internal internal representation of network with $n = 3, H = 3$ and <i>Gaussian</i> activation function trained on the moons dataset	59
3.13	Internal representation of the network with $n = 3, H = 3$ and <i>ReLU</i> as the activation function trained on the moons dataset	60
3.14	Internal representation of network with $n = 3, H = 3$ and <i>tanh</i> as activation function trained on circles dataset	60
3.15	Internal representation of network with $n = 3, H = 3$ and <i>Gaussian</i> activation function trained on the circles dataset	61
3.16	Internal representation of network with $n = 3, H = 3$ and <i>ReLU</i> as the activation function trained on the circles dataset	61
3.17	Node-specific activation of input space for the network with $n = 10, H = 10$ and <i>tanh</i> as the activation function trained on the spirals dataset. Blue denotes activation of -1 , and yellow denotes activation of 1	62
3.18	Node-specific activation of input space for the network with $n = 10, H = 10$ and <i>Gaussian</i> activation function trained on the spirals dataset. Blue denotes activation of 0 , and yellow denotes activation of 1	63
3.19	Node-specific activation of input space for the network with $n = 10, H = 10$ and <i>ReLU</i> as the activation function trained on the spirals dataset.	64

3.20	A comparison of the final hidden layer activation for different architectures with Gaussian activation in example 3: Architecture (a) $n = 10$, $H = 10$ (b) $n = 100$, $H = 3$. Blue regions denote area in the input space where activation of the node becomes 0. Yellow denotes activation of 1. Notice how deeper networks yield complex activations while, shallow-wide networks have relatively simple activations.	65
3.21	(a) Fault space data (b) Fault space classification with a simple input output model	68
3.22	Fault space classification for different activation functions and architectures .	69
4.1	Testing data (blue) and model predictions (orange) for a 5- layer network with 2, 5 and 10 neurons in each hidden layer and σ (Row 1), \tanh (Row 2) and $ReLU$ (Row 3) as activation function	76
4.2	Representation of the absolute function generator with a neural network . .	78
4.3	Triangular wave function	78
4.4	Activation of individual neurons in a network with $ReLU$ (first row) and \tanh (second row) as the activation function and three neurons in the hidden layer	81
4.5	Activation of individual neurons in a deep network with \tanh and $ReLU$ as the activation function and 3 hidden layers	82
4.6	Representation of absolute function by an untrained neural network	83
4.7	Reduced neural network for approximating the absolute function with two parameters	84
4.8	Loss of the reduced network as a function of weights	86
4.9	Trajectory of weights after training with different initializations (green and red circles represent initial weights and trained weights respectively)	86
4.10	Initializations of the reduced network that converge to a global optimum (green) and to a local optimum (red)	87
4.11	(a) Loss of the reduced network corresponding to Equation 4.10 as a function of weights (b) The final layer loss will be uniquely estimated given all the other layers weights and biases	88
4.12	Loss function corresponding to Equation 4.11. It can be seen that regularization makes the function locally convex, and helps the training.	89
4.13	Testing data (blue) and model predictions (orange) for different network configurations for approximating the sinusoidal function with rectified linear unit ($ReLU$) as the activation function (n and L represent the number of nodes in each hidden layer and number of hidden layers respectively)	91
4.14	Testing data (blue) and model predictions (orange) for different network configurations for approximating the sinusoidal function with sigmoid (σ) as the activation function (n and L represent the number of nodes in each hidden layer and number of hidden layers respectively)	92
4.15	Testing data (blue) and model predictions (orange) for different network configurations for approximating the sinusoidal function with hyperbolic tangent (\tanh) as the activation function (n and L represent the number of nodes in each hidden layer and number of hidden layers respectively)	93

4.16	Features of hidden layers and final output of the network approximating the sinusoidal function with 5 hidden layers and 5 nodes per layer with <i>ReLU</i> as the activation function	94
4.17	Features of hidden layers and final output of the network approximating the sinusoidal function with 5 hidden layers and 5 nodes per layer with σ as the activation function	95
4.18	Features of hidden layers and final output of the network approximating the sinusoidal function with 5 hidden layers and 5 nodes per layer with <i>tanh</i> as the activation function	96
4.19	Shekel function described using the parameters in Table 4.1(a) 3D representation of the function approximation task (b) Shows final mapping of the 2D input to the 1D output	98
4.20	Increasing depth (H hidden layers) does not yield increase in performance for a network with 10 nodes per hidden layer	99
4.21	Increasing width (n nodes per layer) of the neural network increases the approximation accuracy of the network with 3 hidden layers	100
4.22	Network comparison: \mathcal{N}_1 and \mathcal{N}_2 . Both have similar performance.	101
4.23	Input region activation of each node in each layer for \mathcal{N}_1 . Each layer plot contains activation of each of the node corresponding to the input space. Yellow is a high value of the activation and blue corresponds to low activation (+1 and -1 respectively) since we are using <i>tanh</i> activation function for each layer	102
4.24	Input region activation of each node in each layer for \mathcal{N}_2 . Each layer plot contains activation of each of the node corresponding to the input space. Yellow is a high value of the activation and blue corresponds to low activation (+1 and -1 respectively) since we are using <i>tanh</i> activation function for each layer	102
4.25	Input region activation of each node in the final layer for \mathcal{N}_1 and \mathcal{N}_2	103
5.1	Architecture of XAI-MEG	107
5.2	Causal graph for a predator prey model with data provided for t , u (Prey fraction), v (Predator fraction). Red corresponds to negative parameters, and blue corresponds to positive parameters. As is seen, $\frac{du}{dt}$ increases with more u , but is negatively impacted due to a second order effect of uv	114
5.3	Simple Harmonic Motion - Inferences by XAI MEG	115
5.4	Damped Simple Harmonic Motion - Inferences by XAI MEG	116
5.5	Lotka Volterra Model- Inferences by XAI MEG	118
5.6	SIR Model - Inferences by XAI MEG	120
5.7	Convection Model - Inferences by XAI MEG	121
5.8	Diffusion Model - Inferences by XAI MEG	122
5.9	Reaction Diffusion Model - Inferences by XAI MEG	124

List of Tables

2.1	Various examples to demonstrate the generation of causal maps using transfer entropy	13
2.2	Description of Tennessee Eastman process variables	16
2.3	List of subsystem-level causal maps	31
3.1	Activation functions and their ranges	41
3.2	The nature of the Loss function exhibiting degeneracy of parameters	56
3.3	Consolidated results of different architectures after 1000 epochs of training .	59
3.4	Fault diagnosis example – Consolidated results after 1000 epochs of training on the test sample	67
4.1	Parameters of the shekel function for the demonstrative example	97
4.2	Depth-Width Effect: Combinations of n and H considered for training shekel function	98

Acknowledgements

*All that is gold does not glitter,
Not all those who wander are lost;
The old that is strong does not wither,
Deep roots are not reached by the frost.*

This thesis is a byproduct of my last 5 years at Columbia University. It would not have been possible without the help and support of a lot of people in my life. While I enter a new phase, I would like to thank a few of the people responsible for this thesis. Those who I may have forgotten, I will say that your contribution is not any lesser.

First I would like to thank my advisor, Prof. Venkat Venkatasubramanian. Prof. Venkat has been a mentor in the truest sense. The vast range of knowledge he has shared with me will forever make me feel indebted to him. He has been the torch bearer for me to shine the light in areas which needed attention, at the same time letting me find my own way – a true Gandalf to my Frodo.

I have had many Sam Wise as well during the course of my study here. I want to thank all my labmates for the really interesting discussions related, and not related to, my work.

Last but not the least, I want to thank my family – My mom, dad, brother, cousins, and my best friend/partner/fiancé. Because of their support these 5 years have been a very

enlightening experience. They did make me find my deepest roots, and I will forever be grateful to them.

Chapter 1: Introduction and Background

The availability of abundant data, powerful hardware, and user friendly software has unleashed an avalanche of machine learning applications in numerous domains [1]. While machine learning is valuable for certain class of problems, it is, however, unable to provide causal explanations or mechanistic insights in general. In this thesis, we look at wide range of problems using methods from pure data-driven techniques all the way to process knowledge driven models, for the purpose of prediction of state of the system and generating mechanistic insights about the process.

We start with the problem of identifying causality between variables in a process system. Causal models are good tools to create explainable systems that show which variables affect the final output variables, thus providing a cause-and-effect chain of transparency and mechanistic insights. Considerable work has gone into creating graphical causal models using data-driven techniques [2, 3] and deriving causal models from equations [4, 5]. While such models analyze the data or the underlying model equations without the inference of the parameters, it often requires the use of a known model form to estimate the parameters from data [6]. Recent work is beginning to address physics-informed machine learning [7, 8, 9], but to truly understand the phenomenology, one requires more transparent machine learning models that exploit domain knowledge for more interpretability [10].

However, we may be able to identify what the hidden features as extracted by neural networks for the purpose of classification and regression. While it is not possible to truly explain the reasons for the underlying structures deep learning models, we attempt to look into the hidden representations of (sufficiently) deep learning models for fault diagnosis and function approximation problems. In recent years, deep neural networks have witnessed great success and widespread acceptance for solving many challenging problems in different

fields. As observed by Venkatasubramanian [1], this explosive growth is due to three key developments: availability of large amounts of data (i.e., "big data"), availability of powerful and cheap hardware, and advances in user-friendly software environments, all of which can be traced directly to the unexpected success of Moore's Law over the last fifty years. While many ideas such as the back propagation algorithm, convolutional neural nets, recurrent neural nets, Bayesian learning, reinforcement learning, etc., have been around for a couple of decades or more, and therefore are not that "new", what is "new", however, is the ease with which these, and other such techniques, can now be applied in a variety of domains due to the three factors mentioned above.

This ease of application of powerful tools has the potential to cause their misuse and abuse, if one is not careful. This concern is similar to what we have witnessed over the years with statistics [11]. In this regard, the current state of machine learning is akin to that of alchemy before chemistry and chemical engineering, involving a lot of guess-and-test trials, as noted by Jordan [12] and Hutson [13].

The vast majority of recent papers on machine learning, particularly in chemical engineering, is aimed at employing a deep neural network as a tool for solving some application, and not at developing a deeper understanding as to how and why the network derives its problem-solving capability. These papers seem to reflect an attitude that "Well, this particular deep neural network that we specifically crafted after many guess-and-test trials seem to work well enough for this particular application we are interested in, and we don't really care why and how it does so! It does the job. For the next application, we will try something else along these lines, and, hopefully, that will do its job as well!".

Important questions such as what kind of neural network to use, how many hidden layers, how many neurons in each layer, what kind of activation or squashing function to use, how to initialize the weights, and on and on, are largely determined by an Edisonian guess-and-test fashion that reminds one of alchemy more than of chemistry, as noted. A systematic approach to answering these design questions is largely absent in many applications, and such decisions

are highly dependent on the expertise (or the lack thereof) of the programmer. Despite the seeming practical success of deep neural nets, we find this state of affairs deeply unsatisfactory for at least three reasons. First, it reveals our lack of fundamental understanding of the theory of deep neural networks. Second, this makes the use of neural networks in some engineering applications worrisome due to their lack of generalizability and to their potential exposure to adversarial attacks. Such drawbacks are perhaps not very serious in recognizing cats vs dogs, or in recommendation systems (e.g., Yelp, Rotten Tomatoes, etc.), but can be quite important in applications such as fault diagnosis, process safety analysis, where the cost of a mistake could be quite high. Third, there is no causal explanation provided which is needed in many engineering applications.

How do we incorporate mechanistic knowledge in these models? It is shown that deep neural networks have been shown to have great degeneracy in their feature extraction performance, and hence are not easily interpretable in general [14, 15]. The causal explanations are important in many science and engineering applications that are governed by fundamental principles of physics, chemistry, and/or biology. In this regard, these areas are different from applications such as game playing, computer vision, and recommendation systems, where there are no such conservation laws or constitutive equations governing their workings. Thus, it makes sense in those domains to be mostly data-driven.

However, our applications are quite different, and we often would like to understand from first principles why a particular decision was reached (or not reached) by the machine learning system. This is particularly important for mission-critical applications such as process control, fault diagnosis, and process safety analysis, where the cost of a mistake could be potentially quite high in comparison with recommendation systems such as Yelp or Rotten Tomatoes. If one has a bad experience at a restaurant recommended by Yelp, one might lose a couple of hours and some money. However, if an intelligent control system makes a mistake in the domains of chemical engineering, aeronautical engineering, or nuclear engineering, it could cost lives. In such applications we would prefer not to rely on purely

black-box models.

To circumvent the issue of model intractability and explainability, much work has been done in creating sparse models, under the constraint that only a few modes are active during the course of the process [16, 17]. This typically involves use of a high dimensional feature representation, which when combined with sparse optimization results in models where only few features are active. This is often used for reduced order modeling. However, these models are driven by identifying higher-order nonlinear features from data, followed by an estimation of the parameters of the model. A key missing element in such identification is the lack of leveraging of a priori first-principles knowledge to specify the allowed functional transformations of variables and their combinations [18, 19]. Further, the use of these features and parameters to generate a physical explanation of the combinations is often not addressed. To avoid these use cases, we finally show how causal explanations can be generated for physicochemical systems, where the underlying model forms may be known to a reasonable degree, however, the true modes of operations are unknown. Machine learning is used to identify these underlying causal model forms generated through mechanistic knowledge about physicochemical systems.

Chapter 2: From data to causal models

Cause-and-effect reasoning is at the core of fault diagnosis and hazards analysis in process systems, thereby requiring the development and use of causal models for automated approaches. Furthermore, causal models are also required to explain the decisions and recommendations of artificial intelligence-based systems, lack of which is a serious drawback of purely data-driven approaches. Here, we demonstrate an approach for building multi-level causal models. A hierarchical approach is proposed to capture both cyclic and non-cyclic features of a process plant. Decoupling these features of the plant by constructing two tiers of digraphs, one tier representing overall plant and the other representing individual subsystems, helps in better inference of causal relations present in the system. An algorithm that subsides the effects of indirect causal interactions using reachability matrix and adjacency matrix ideas is also proposed. The algorithm is tested on the Tennessee Eastman benchmark process and the resulting causal model is found to represent the true causal interactions present in the system.

Causality has been a long debated topic amongst philosophers, and in the last quarter of a century with the advent of artificial intelligence (AI), within the scientific community as well [20]. AI-based systems that rely solely on purely data-driven approaches often lack the ability to provide causal explanations of their recommendations. A deep neural network, for example, might successfully learn to classify different faults in a process plant from real-time data, and correctly identify them when they occur, but it typically would not be able to explain how it arrived at its decision. For instance, it might correctly identify valve failure in a coolant line as the cause of the high temperature alarm in a reactor, but it wouldn't be able to provide the causal chain of how the valve failure led to the high temperature alarm. The neural net might be able to answer the question of what caused the temperature alarm

but not why or how. It also cannot answer why not some other failure as the root cause.

To answer such questions, to generate step-by-step cause-and-effect explanations, one needs a causal model of the system. This work is about the development of such models from real-time process data. The causal models are represented in the form of directed graphs, a widely used tool for diagnosis and safety analysis, which model the directionality of the cause-effect relationship between variables [21, 22, 23]. This graph theoretic representation with nodes and edges is termed as a causal map or causal graph¹. There is considerable literature on this topic, particularly towards developing a mathematical framework for causality [24, 25, 26, 27, 28]. One of the most prevalent measures of causality is Granger causality [25], which has gained popularity in recent decades. The notion that cause(s) should always precede the effect(s) suggests that the past of the cause(s) contains information about the future, the effect(s). In his definition, Granger said that a variable, x is said to cause another y , if knowing the past of x increases the predictability of the future of y . Though initially applied in a linear setting on time-series from an economics perspective, the metric and its extensions have shown promise in biological sciences [29]. For nonlinear interactions, transfer entropy as a causality measure has shown a lot of potential in chemical engineering and biological examples [30, 31, 32, 33]. For a Gaussian system, both transfer entropy and Granger causality are the same [34].

Though these different causality measures try to ascertain the cause-effect relationships from data, they are in fact making causal inferences from correlations in the data. One of the major issues with these causality measures is that the correlations between variables could be a result of direct causation, or causation through an intermediary variable. Existing causality measures are incapable of distinguishing between direct and indirect causations. This issue of indirect causation is more pronounced in systems with cycles due to cyclic information transfer. In presence of cycles, every variable would seem to affect every other variable due to the presence of indirect paths between them. In such systems, use of existing

¹The terms 'causal graph' and 'causal map' are used interchangeably in this article.

algorithms/metrics would result in a causal model that does not differentiate true causes from indirect causes. Take the example of a control loop. At a finer level of granularity, we can think of every variable in the loop to causally affect every other variable (due to indirect causation). However, at a coarser level, the causal information is that the set-point is the cause and the process variable is the effect. From a utilitarian standpoint, the information from this coarser level of granularity is more useful while the information that every variable causes every other variable is not of much use. Knowledge about the system of interest would be beneficial to avoid such inferences, but this information is typically hard to obtain for most systems (e.g. biological systems).

However, for most chemical process systems, valuable information in the form of flowsheets, first-principles models, etc., is available. While one can extract causal models using only data-driven techniques, such as transfer entropy, ignoring all this valuable information seems to be an unwise choice. It will be helpful to augment the causal model generated from data with such additional information as that would make the resultant causal model more reliable. This is particularly important because most process plants have control loops and/or recycles, and hence the blind use of causality measures without contextual information could result in inaccurate causal models. Our desire in this paper is to avoid such pitfalls and to develop a hybrid framework by integrating data-driven and flowsheet-derived causal information.

Though there are various approaches for developing causal maps using data-driven [30, 31] and process model-based methods[35, 36, 37], our proposed method is a hybrid that combines data-driven causality with structural information from the flowsheet. The causal map developed, however, varies for different length scales of the process as explained before in the case of a simple control loop. It is essential to work at these different length scales and represent the causal map at these different scales to gain more insight into the causal structure of the system. A plant-level and subsystem-level hierarchy is set up using information from the process flowsheet, by choosing variables corresponding to the two levels of granularity.

This approach is key for this work as we generate causal maps at two levels – (i) an acyclic map at plant-level and (ii) a cyclic map at subsystem-level. Since the plant-level graph is an acyclic system, the resultant causal graph must be a directed acyclic graph (DAG)². The subsystem-level causal map is a cyclic graph as cyclic effects due to the presence of control loops and recycles are included. We use transfer entropy as the measure of causality in this article for the purpose of demonstration as it is a reasonable metric for non-linear systems [30, 33].

This paper is organized as follows – section 2.1 explains the methodology of constructing causal maps using transfer entropy calculated from real-time data. The method is demonstrated using various examples. In the following section (section 2.2), we describe the well-known Tennessee Eastman process which is used as a testbed. Section 2.3 describes our hierarchical approach to analyze large scale systems and the graph reduction strategy to obtain a better measure of direct causality. This graph reduction strategy is only applicable for graphs without cycles, and hence can only be applied to plant level graph. The overall method is tested on the Tennessee Eastman benchmark process described in section 2.2.

2.1 Transfer Entropy as a Measure of Causality

Although the hierarchical approach proposed in this article can work with any causality metric, transfer entropy is used for quantifying causality for the purpose of demonstration. Transfer entropy is an information-theoretic measure introduced by [27] that extracts the amount of directed transfer of information from one variable (x) to another (y) and is defined as

$$t_{x \rightarrow y} = \sum_i p(y_{i+h}, \mathbf{y}_i, \mathbf{x}_i) \cdot \log \left(\frac{p(y_{i+h} | \mathbf{y}_i, \mathbf{x}_i)}{p(y_{i+h} | \mathbf{y}_i)} \right) \quad (2.1)$$

where $\mathbf{x}_i = [x_i, x_{i-\tau}, \dots, x_{i-(k-1)\tau}]$ and $\mathbf{y}_i = [y_i, y_{i-\tau}, \dots, y_{i-(l-1)\tau}]$ are the embedded vectors. k and l are the embedding dimensions of x and y respectively. The prediction horizon is h and

²DAG is a directed graph with a topological ordering such that there is no path from one vertex/node to itself.

τ is the time interval considered between various variables in the embedding vectors of x and y . $p(\cdot, \cdot)$ represents the joint probability distribution function (PDF) while $p(\cdot|\cdot)$ represents the conditional probability or transitional probability. Joint PDFs can be estimated using Kernel method, details of which can be found elsewhere [27, 30]. The values of h and τ in our case were estimated using the lag between the time-series when the value of the cross-correlation was the highest. $k = 2$ and $l = 1$ are chosen for estimation of transfer entropy throughout this manuscript.

It can be easily seen from Equation (2.1) that if y is independent of x , then $p(y_{i+h}|\mathbf{y}_i, \mathbf{x}_i)$ will be equal to $p(y_{i+h}|\mathbf{y}_i)$ and hence, $t_{x \rightarrow y}$ will be zero indicating that no information is transferred from x to y [27, 30]. As measured data is usually corrupted with noise, we may not obtain zero transfer entropy for a non-causal pair. Hence, we have to make a judgment to assess whether the value obtained is significant or not. Given the time-series data of x and y , to determine whether x causes y or, in other words, to determine whether there exists an edge from x to y in the causal map, a hypothesis test on the transfer entropy value is used. This is represented as follows:

$$\text{Null hypothesis, } H_0 : t_{x \rightarrow y} = 0 \quad (2.2)$$

$$\text{Alternate hypothesis, } H_a : t_{x \rightarrow y} > 0 \quad (2.3)$$

[38] proposed a Monte Carlo method using surrogate data sets for hypothesis testing. Different algorithms are available for generating surrogate data sets and we use the iterative amplitude adjusted Fourier Transform (iAAFT) technique proposed by [39] for the same. The principle behind the Monte Carlo-based hypothesis testing is that the surrogate data sets generated do not have any causal relationship between the variables and hence should have zero transfer entropy value. Now, assuming that the transfer entropy values of the surrogate data sets ($t_{x \rightarrow y}^{surr, k} \quad \forall k \in \{1, \dots, N_s\}$ where N_s is the total number of surrogate pairs) fall in a bell-curve, the null hypothesis can be rejected if the transfer entropy of the

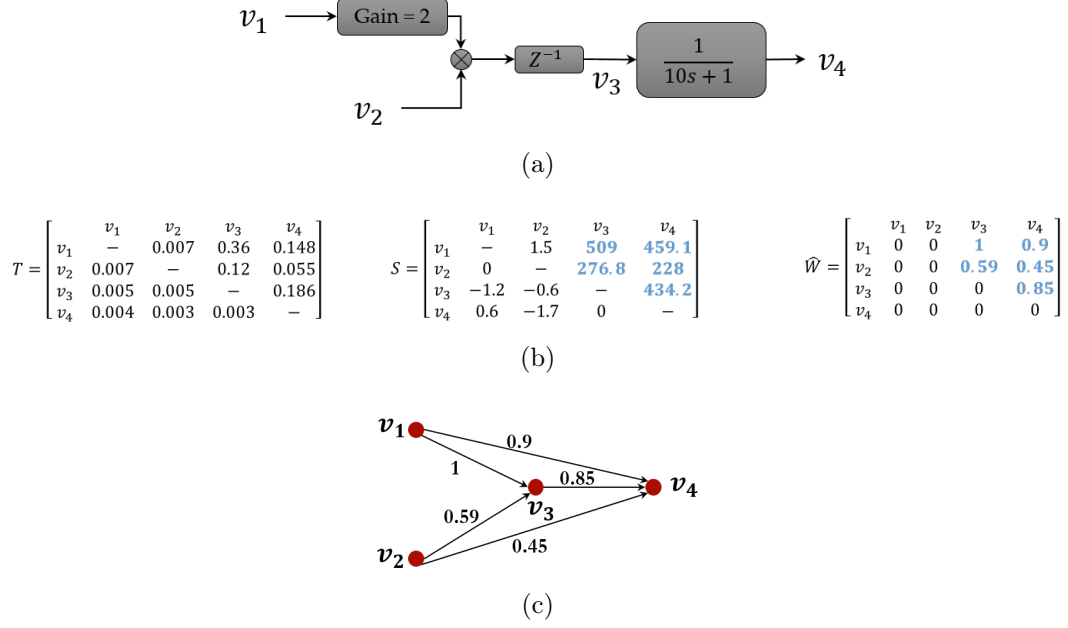


Figure 2.1: (a) A simple case study of mixer simulated in MATLAB simulink (b) Significance value matrix and the derived weighted adjacency matrix. Weights greater than threshold are marked in blue. (c) Transfer entropy-based digraph generated for the system

true data is such that

$$\text{Significance value, } s_{x \rightarrow y} > s_0 \quad (2.4)$$

$$\text{where } s_{x \rightarrow y} = \frac{t_{x \rightarrow y} - \mu_{t_{x \rightarrow y}}^{surr}}{\sigma_{t_{x \rightarrow y}}^{surr}} \quad (2.5)$$

where $\mu_{t_{x \rightarrow y}}^{surr}$ and $\sigma_{t_{x \rightarrow y}}^{surr}$ are the mean and standard deviation of the distribution corresponding to the transfer entropy values of the surrogate pairs, and s_0 is the threshold that defines the boundary. Above equation suggests that x causes y if their transfer entropy ($t_{x \rightarrow y}$) is far from the distribution of their surrogate-pair transfer entropies. Higher the value of significance for a pair, more prominent is the causal interaction between them. For this reason, significance value is an indication of strength of the connection between the variables. Constructing digraph based on these significance values is discussed next.

2.1.1 Generating digraph based on transfer entropy

Let us consider a process with N variables represented by $x^{(j)}$ for $j \in \{1, \dots, N\}$. Let T be an $N \times N$ matrix such that $T(i, j)$, or simply T_{ij} , represents $t_{x^{(i)} \rightarrow x^{(j)}}$. Similarly, let S represent the matrix of significance values such that $S(i, j)$ (or S_{ij}) represents $s_{x^{(i)} \rightarrow x^{(j)}}$. The adjacency matrix estimated from the data is given by,

$$\hat{A}(i, j) = \hat{A}_{ij} = \begin{cases} 1 & \text{if } s_{x^{(i)} \rightarrow x^{(j)}} \geq s_0 \\ 0 & \text{if } s_{x^{(i)} \rightarrow x^{(j)}} < s_0 \end{cases} \quad (2.6)$$

Similarly, a corresponding weighted adjacency matrix (\hat{W}) can be constructed as follows:

$$\hat{W}(i, j) = \hat{W}_{ij} = \begin{cases} \frac{s_{x^{(i)} \rightarrow x^{(j)}}}{s^{max}} & \text{if } \frac{s_{x^{(i)} \rightarrow x^{(j)}}}{s^{max}} \geq w_0 \\ 0 & \text{if } \frac{s_{x^{(i)} \rightarrow x^{(j)}}}{s^{max}} < w_0 \end{cases} \quad (2.7)$$

where s^{max} is the maximum positive significance value obtained. This weighted adjacency matrix shows the relative importance of an edge with respect to other edges in the graph. A higher value of $\hat{W}(i, j)$ represents a stronger edge from i to j . Causal maps can be constructed based on this weighted adjacency matrix. If $w_0 = 0.05$, it means that edges having a significance value of at least 5% of the maximum significance value (s^{max}) are considered for constructing the causal map.

To demonstrate the importance of this technique, a simple case study is considered. A mixer as shown in Figure 2.1(a) is simulated using MATLAB Simulink and the data obtained is used to calculate transfer entropy. By generating 20 surrogate data sets, significance value is estimated for each pair. Transfer entropy matrix, significance value matrix and the corresponding weighted adjacency matrix are provided in Figure 2.1(b). Diagonal elements are not evaluated as we assume that self-loops are not present in the system. The directed graph or causal map obtained using this procedure is shown in Figure 2.1(c). Note that the graph obtained captures all significant causal interactions including indirect effects.

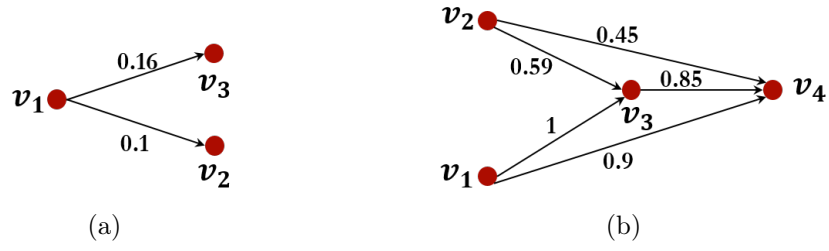


Figure 2.2: Causal maps (a) and (b) are obtained based on equation (2.8) for examples (c) and (d) in Table 2.1 respectively. It can be seen that many of the significant connections are missing. The reason can be attributed to the presence of feedback loops/recycle streams.

Table 2.1: Various examples to demonstrate the generation of causal maps using transfer entropy

Example No.	System	Weighted Adjacency Matrix	Causal Map
(a)		$\hat{W} = \begin{bmatrix} & v_1 & v_2 & v_3 & v_4 \\ v_1 & - & 0 & 1 & 0.9 \\ v_2 & 0 & - & 0.59 & 0.45 \\ v_3 & 0 & 0 & - & 0.85 \\ v_4 & 0 & 0 & 0 & - \end{bmatrix}$	
(b)		$\hat{W} = \begin{bmatrix} & v_1 & v_2 & v_3 & v_4 \\ v_1 & - & 0 & 0.04 & 0.03 \\ v_2 & 0 & - & 1 & 0.87 \\ v_3 & 0 & 0 & - & 0.61 \\ v_4 & 0 & 0 & 0 & - \end{bmatrix}$	
(c)		$\hat{W} = \begin{bmatrix} & v_1 & v_2 & v_3 \\ v_1 & - & 1 & 0.09 \\ v_2 & 0 & - & 0.07 \\ v_3 & 0 & 0.1 & - \end{bmatrix}$	
(d)		$\hat{W} = \begin{bmatrix} & v_1 & v_2 & v_3 & v_4 \\ v_1 & - & 0.26 & 0.13 & 0.60 \\ v_2 & 0.06 & - & 0.10 & 0.20 \\ v_3 & 0.15 & 1 & - & 0.30 \\ v_4 & 0.22 & 0.17 & 0.35 & - \end{bmatrix}$	

The causal map obtained using this approach is capable of incorporating bidirectional edges, and thus can detect all possible interactions in the presence of cycles. Table 2.1 shows various results obtained using this approach for different simple systems simulated in MATLAB Simulink. Examples (a) and (b) shows that the technique is able to capture changes in system properties. The value of 'Gain' was changed from 2 to 0.02 from example (a) to (b) and the weighted adjacency matrix was able to capture that effect. It can be seen that the strength of the edges from v_1 to both v_3 and v_4 reduced considerably from (a) to (b). This shows the potential of this metric in fault diagnosis applications. Examples (c) and (d) shows the capability of the algorithm to capture bidirectional edges.

If it is known a priori that the system under consideration does not contain any bidirectional edges or if we are interested in constructing a DAG, we can enforce this constrain on the graph by comparing the values of $s_{x(i) \rightarrow x(j)}$ and $s_{x(j) \rightarrow x(i)}$ for every pair of variables and use the edge with higher weight to construct the graph. The weighted adjacency matrix for such a graph can be constructed as follows:

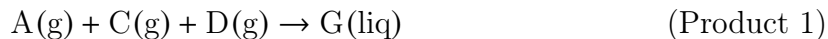
$$\hat{W}_{ij} = \begin{cases} w_{ij} & \text{if } w_{ij} \geq w_0 \\ 0 & \text{if } w_{ij} < w_0 \end{cases} \quad (2.8)$$

where $w_{ij} = \frac{\Delta s(i, j)}{\Delta s^{max}}$, $\Delta s(i, j) = s_{x(i) \rightarrow x(j)} - s_{x(j) \rightarrow x(i)}$ and $\Delta s^{max} = \max(\Delta s(\cdot, \cdot))$. The \hat{W} matrix obtained can then be used to construct the corresponding causal map. This approach is similar to the one presented in [30] except that here the difference of significance values is considered unlike the difference of transfer entropies in [30]. Although this method works well for systems that can be represented as a DAG, this approach fails to capture many significant connections for systems with feedback or recycles. For example, Figures 2.2(a) and 2.2(b) show the causal maps obtained using equation (2.8) for examples (c) and (d) in Table 2.1. In Figure 2.2(a), direct interactions between v_2 and v_3 are not captured. Similarly, in Figure 2.2(b), direct interactions between v_1 and v_2 are missing. Fault diagnosis will be highly biased if we rely on these causal maps instead of the map obtained for systems with cyclic features. Hence, it is mandatory to ensure that feedback loops and recycle streams are not present before applying Equation (2.8). Note that if a different measure of causality is used (instead of transfer entropy), then a weighted adjacency matrix can be generated using that metric using the same strategy as explained in this section.

2.2 Tennessee Eastman Benchmark Process

For the purpose of demonstration on large-scale process industries, the well studied Tennessee Eastman benchmark process is considered. This process involves the production of

G and H from the reactants A, C, D and E. In addition to these 6 components, there is an inert B and a byproduct F associated with the process. The reactions are as follows;



The process has 5 major units: a reactor, a condenser, a product separator, a compressor and a product stripper. Table 2.2 shows the list of 41 measured variables and 12 manipulated variables. Further details of the process can be found elsewhere [40, 41, 42, 43]. Various control strategies can be applied for this process as discussed in [43, 42]. We consider the decentralized control strategy presented in [43] for our analysis and the corresponding process flowsheet including these control loops is shown in Figure 2.3. Details of the controlled and manipulated variables in the 19 control loops (or 8 multi-level control loops) present in this flowsheet can be found in [43]. The MATLAB codes and Simulink models for the simulation of this process can be found at <http://depts.washington.edu/control/LARRY/TE/download.html>. Data collected using the Simulink model for the base case scenario mentioned in [43] is used for analysis. Figure 2.4 shows the dynamic profiles of variables 1, 9, 13 and 20. The data is standardized by dividing the mean-centered data with its standard deviation before evaluating transfer entropies.

Table 2.2: Description of Tennessee Eastman process variables

Legend	Type	Description	Legend	Type	Description
v_1	Measurement	A feed (Stream 1)	v_{28}	Measurement	Composition of F (Stream 6)
v_2	Measurement	D feed (Stream 2)	v_{29}	Measurement	Composition of A (Stream 9)
v_3	Measurement	E feed (Stream 3)	v_{30}	Measurement	Composition of B (Stream 9)
v_4	Measurement	A and C feed (Stream 4)	v_{31}	Measurement	Composition of C (Stream 9)
v_5	Measurement	Recycle flow (Stream 8)	v_{32}	Measurement	Composition of D (Stream 9)
v_6	Measurement	Reactor feed rate (Stream 6)	v_{33}	Measurement	Composition of E (Stream 9)
v_7	Measurement	Reactor pressure	v_{34}	Measurement	Composition of F (Stream 9)
v_8	Measurement	Reactor level	v_{35}	Measurement	Composition of G (Stream 9)
v_9	Measurement	Reactor temperature	v_{36}	Measurement	Composition of H (Stream 9)
v_{10}	Measurement	Purge rate (Stream 9)	v_{37}	Measurement	Composition of D (Stream 11)
v_{11}	Measurement	Product separator temperature	v_{38}	Measurement	Composition of E (Stream 11)
v_{12}	Measurement	Product separator level	v_{39}	Measurement	Composition of F (Stream 11)
v_{13}	Measurement	Product separator pressure	v_{40}	Measurement	Composition of G (Stream 11)
v_{14}	Measurement	Product separator underflow (Stream 10)	v_{41}	Measurement	Composition of H (Stream 11)
v_{15}	Measurement	Stripper level	v_{42}	Manipulated	D feed flow (Stream 2)
v_{16}	Measurement	Stripper pressure	v_{43}	Manipulated	E feed flow (Stream 3)
v_{17}	Measurement	Stripper underflow (Stream 11)	v_{44}	Manipulated	A feed flow (Stream 1)
v_{18}	Measurement	Stripper temperature	v_{45}	Manipulated	A and C feed flow (Stream 4)
v_{19}	Measurement	Stripper steam flow	v_{46}	Manipulated	Compressor recycle valve
v_{20}	Measurement	Compressor work	v_{47}	Manipulated	Purge valve (Stream 9)
v_{21}	Measurement	Reactor cooling water outlet temperature	v_{48}	Manipulated	Separator pot liquid flow (Stream 10)
v_{22}	Measurement	Separator cooling water outlet temperature	v_{49}	Manipulated	Stripper liquid product flow (Stream 11)
v_{23}	Measurement	Composition of A (Stream 6)	v_{50}	Manipulated	Stripper steam valve
v_{24}	Measurement	Composition of B (Stream 6)	v_{51}	Manipulated	Reactor cooling water flow
v_{25}	Measurement	Composition of C (Stream 6)	v_{52}	Manipulated	Condenser cooling water flow
v_{26}	Measurement	Composition of D (Stream 6)	v_{53}	Manipulated	Agitator speed
v_{27}	Measurement	Composition of E (Stream 6)			

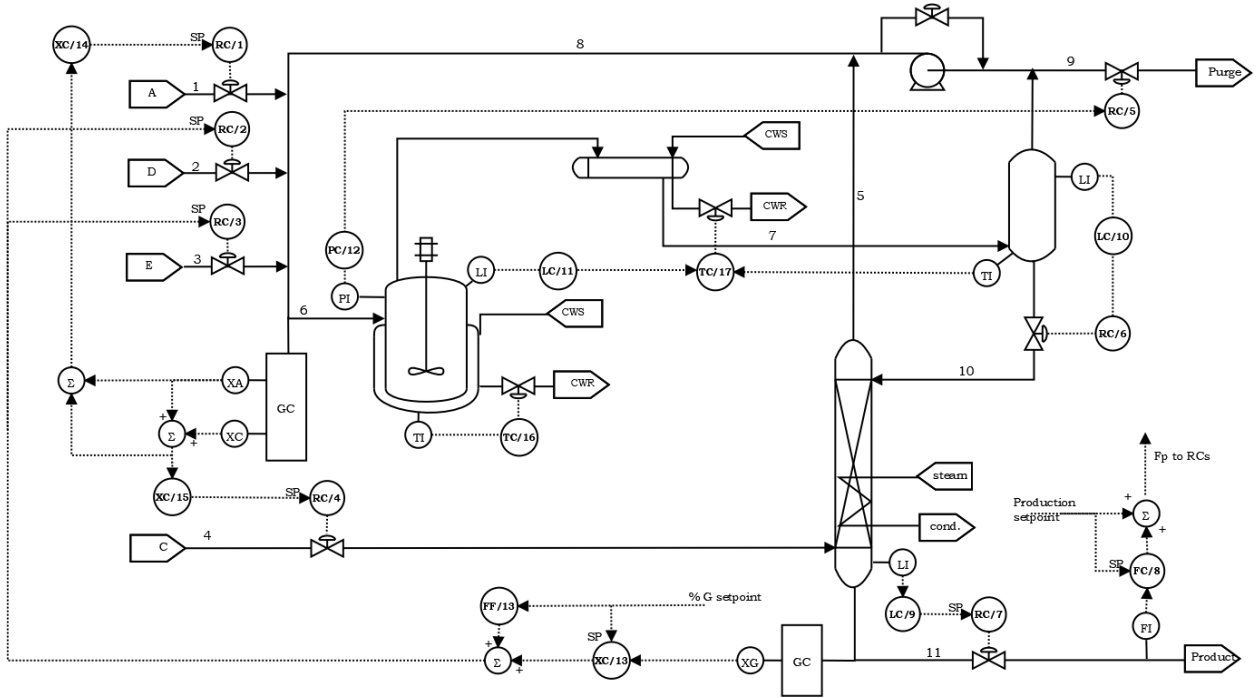
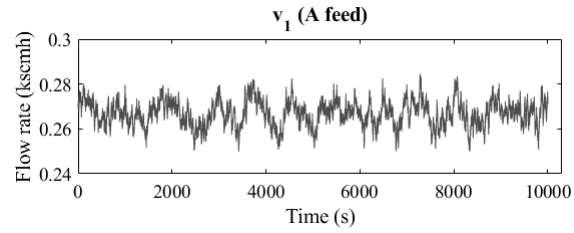


Figure 2.3: Process flowsheet of the Tennessee Eastman benchmark process, with a decentralized control scheme (As seen in [43])

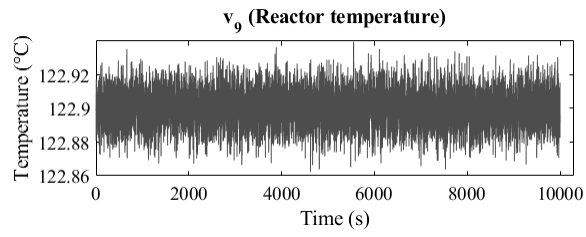
2.3 Hierarchical framework for developing causal maps

Data-driven causality detection algorithms, in general, lack the ability to differentiate between direct and indirect causality. The effect of this inability becomes more pronounced when we are dealing with a large-scale chemical plant which has a large number of units. The presence of control loops and recycle streams aggravates the task at hand. When we rely on process data for detecting causality in such a plant, it is likely to obtain an intricate graph that contains many spurious edges due to indirect causality. The value of such a graph would be limited in terms of fault diagnosis or root cause analysis.

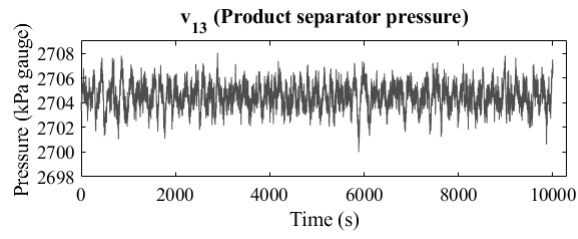
To deal with the complexity of these large-scale systems, we propose a hierarchical approach where the variables related to control loops or recycle streams are decoupled from the remaining variables. The resulting causal map contains 2 tiers: first one corresponding to the whole plant while the second one corresponding to individual units or control loops (subsystems). This segregation helps us analyze the system at two levels of granularity. Hierarchical approach requires us to manually choose the variables for each causal map based on the information from flowsheet. The significance of this approach is that at the plant-level, the causal map would be a directed acyclic graph (DAG) and thus, fault identification is more straightforward. By comparing plant-level causal map at any particular time with that of the desired plant-level causal map (reference map) and tracking changes in the edge weights, we expect to identify the fault propagation path and the root-cause. Once a possible root cause is identified from the plant-level, we can obtain more insights regarding the fault by looking at the subsystem-level causal map related to the isolated variable (if a subsystem-level map exists related to that variable). For example, if the plant-level causal map identifies that reactor temperature is the root cause, then by probing further into the causal map of the control loop that manipulates the coolant flow rate to control the reactor temperature, it can be seen whether the fault is with the controller or the valve or the coolant temperature. Fault diagnosis using this 2-tier causal map is outside the scope of this article.



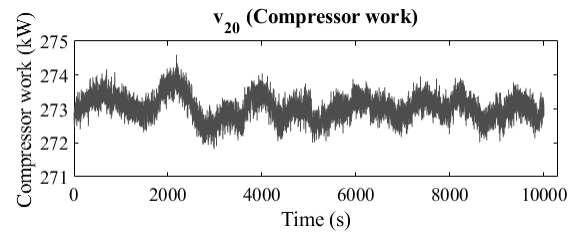
(a)



(b)



(c)



(d)

Figure 2.4: Dynamic profiles of variables 1, 9, 13 and 20.

2.3.1 Tier 1: Plant-level DAG

A subset of variables is chosen from the variables of interest such that the resulting causal map has minimal cyclic interactions so that we can obtain a DAG for this subset of variables. Variable selection is performed based on prior knowledge of the process in the form of a process flowsheet. All variables that would directly result in cycles or bidirectional edges in the causal map are omitted for plant-wide analysis. Following are a few rules for choosing the subset of variables for plant-level analysis (Note that these rules may not be sufficient to obtain DAG for all processes. Variable selection must be carefully performed for the process under consideration such that the subset of variables form a DAG based on the flowsheet (prior) information).

- Rule 1: For control loops, only controlled variables are chosen. Manipulated variables, controller outputs etc. are omitted (refer Figure 2.5(a)) to avoid possible cycles in the causal map. Further, this helps us look at control loop interactions with other variables of interest.
- Rule 2: For units like reactors, distillation columns etc., operating conditions like temperature and pressure of the unit are always chosen to ensure that the effects of operating conditions on other variables are captured in the causal map.
- Rule 3: If an output of a unit is fed back to itself as a recycle stream, the overall system along with the recycle stream is considered as a subsystem as shown in Figure 2.5(b), and the input and output streams of this overall system are chosen for plant-level analysis. Otherwise, there will be cycles in the resulting causal map as the recycle stream (stream 4 in Figure 2.5(b)) and input stream (stream 2 in Figure 2.5(b)) are inter-related.
- Rule 4: For each unit, properties of its input and output streams are considered except when that stream acts as a recycle stream to another unit. For example, in Figure 2.5(c), all properties of streams 2 and 3 are included while properties of stream 4 and 5 are

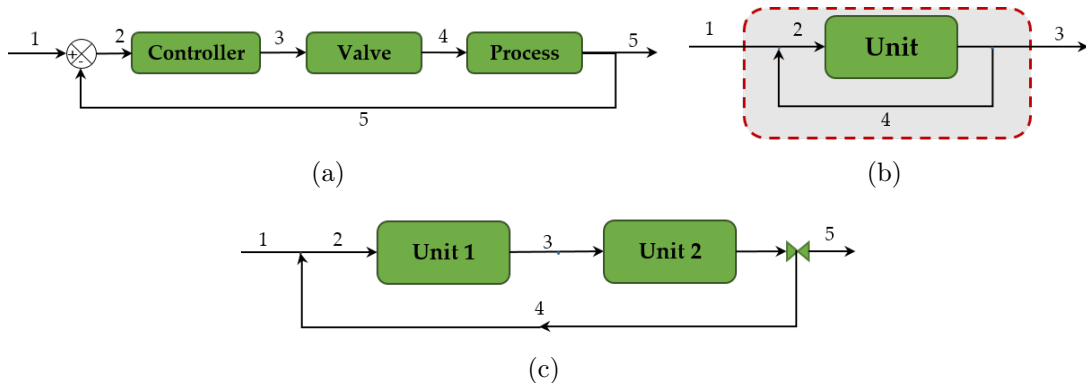


Figure 2.5: (a) Example for a feedback system. All streams except 5 are omitted for analysis. (b) Example for a unit with recycle stream. The subsystem marked by the red dotted line is considered as a single unit and thus, only streams 1 and 3 are included for developing plant-level DAG. (c) Example for the case where 2 units are connected using a recycle stream.

omitted. Again, if streams 2 (or 3) and 4 are considered together, there will be cycles in the causal map as stream 4 acts as a recycle stream to unit 1. Also, stream 1 can be considered as far as it doesn't violate any other rules.

Rule 5: If multiple levels of controllers are used, the controlled variable corresponding to the master controller (higher level controller) alone is included for analysis. This would eliminate possible bidirectional edges or cycles due to dependent control loops.

It might seem that these rules restrict the causal structure present in the system. Since the goal of this section is to obtain an overall causal structure, the rules provide a heuristic for causal inference based on knowledge about the system, at the plant-level. It might be possible that there exist a few bidirectional causal interactions between the chosen variables. To obtain a DAG, we restrict the presence of such bidirectional edges by picking the one that's more significant among the two. For example, as properties (say temperature) of all units are considered for analysis, it is possible that temperatures of two different units affect each other because of the presence of a recycle stream between these units. However, the connection that's more significant would be chosen to construct the plant-level DAG by using equation (2.8).

For the Tennessee Eastman case study discussed in section 2.2, 22 variables (out of 41

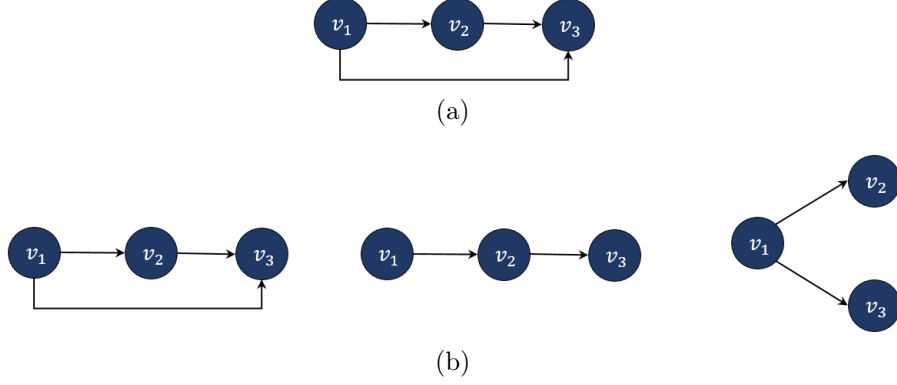


Figure 2.6: Demerits of data-driven causality detection algorithms. If a data-driven algorithm results in a causal map as shown in (a) for a system S , its true representation could be any of the causal maps shown in (b).

measured variables) are shortlisted such that there are minimal cyclic interactions among them and are used for the construction of plant-wide digraph. Variables chosen include 1 flow rate (v_6), 3 levels (v_8, v_{12}, v_{15}), 4 temperatures ($v_9, v_{18}, v_{21}, v_{22}$), 3 pressures (v_7, v_{13} and v_{16}) and 11 concentrations (v_{23} to v_{28} and v_{37} to v_{41}). Other variables are neglected either because they are manipulated variables or because they are associated with recycle streams.

Figure 2.7 shows the digraph obtained for this subset of variables. A total of 60 edges are obtained for $w_0 = 0.12$. As our aim was to develop a DAG for the plant, we have constrained the presence of self loops and bidirectional edges, and the variables were shortlisted such that there are no direct cyclic interactions among them. However, it can be seen that although the obtained graph does not have any bidirectional edges (or cycles of length 2) or self-loops, it contains cycles of path length more than 2 (path length here refers to the number of edges that constitute the cycle). For example, the path $v_{16} \rightarrow v_{24} \rightarrow v_{25} \rightarrow v_{16}$ is a cycle present in this causal map. These cycles could be due to the noise present in the data used for calculations, due to the presence of indirect edges or due to interdependencies between units (while all unit properties are considered).

Furthermore, by comparing the graph and the actual flowsheet of the process, it is evident that many of the edges obtained are due to indirect causation. If a variable v_1 affects a vari-

Observed Graph (based on \hat{W})

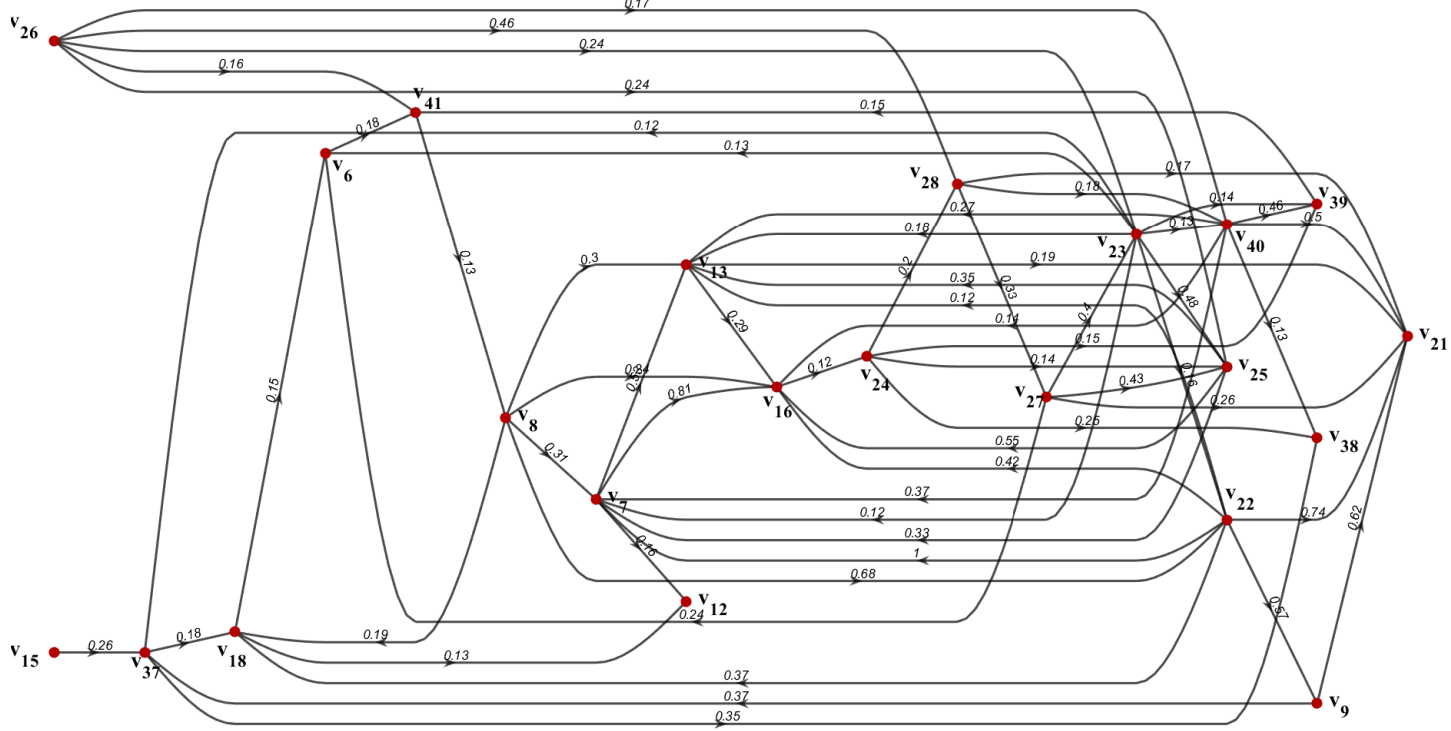


Figure 2.7: Plant-level causal map constructed based on transfer entropy for the Tennessee Eastman benchmark process (Base case). $w_0 = 0.12$ is used as threshold.

able v_2 which in turn affects another variable v_3 , then the data is likely to show a dependency from v_1 to v_3 . But it is also possible that there is in fact a direct causation from v_1 to v_3 . As a matter of fact, if we observe a causal map as shown in Figure 2.6(a) using a data-driven algorithm, true representation of that system could be any of the causal graphs shown in Figure 2.6(b). It is very difficult to distinguish between direct and indirect edges from what you observe unless we have some prior knowledge about the system. Various researchers have looked into techniques for the reduction of graphs obtained from data-driven approaches by identifying indirect causations. In the approach adopted by [30], whenever there exists an alternate path between two variables (say v_1 and v_3) using intermediate/confounding variables

(like v_2 in this example), the graph was reduced by completely neglecting the possibility of a direct path from v_1 to v_3 . There are various other techniques based on partial correlation [25], partial association [44], partial directed coherence [45], partial transfer entropy [46], direct transfer entropy [31] etc. that evaluate the causal relation from v_1 to v_3 by conditioning on v_2 to identify direct causality. Though these approaches may be effective in differentiating direct and indirect causality, it becomes impractical and computationally prohibitive when we are dealing with large number of variables. There is also a possibility of more than one intermediary variable in which case the number of evaluations necessary for the reduction to direct causal graph would be enormous.

Plant-level graph reduction strategies

In this section, we will discuss simple strategies proposed to remove cycles and strengthen the direct edges present in the causal map obtained from transfer entropy. These strategies are applicable only for systems with a DAG representation. This technique is based on the properties of reachability matrix [47] and powers of adjacency matrix. The following description of this property is adapted from [48]:

A path in a network is any sequence of vertices such that every consecutive pair of vertices in the sequence is connected by an edge in the network. The length of a path in a network is the number of edges traversed along the path (not the number of vertices). For either a directed or an undirected graph the element A_{ij} is 1 if there is an edge from vertex i to j , and 0 otherwise. Then the product $A_{ik}A_{kj}$ is 1 if there is a path of length 2 from j to i via k , and 0 otherwise. And the total number $N_{ij}^{(2)}$ of paths of length two from i to j , via any other vertex, is $N_{ij}^{(2)} = [A^2]_{ij}$ where $[...]_{ij}$ denotes the ij^{th} element of a matrix. Generalizing to paths of arbitrary length r , we see that $N_{ij}^{(r)} = [A^r]_{ij}$.

This property can be extended to weighted adjacency matrices where we know the weights of each edge rather than simply the information about the presence/absence of edges. Sup-

pose the matrix W^* represents the true weighted adjacency matrix of a process such that its ij^{th} element, W_{ij}^* , is the weight corresponding to a direct edge from variable i to j . Higher powers of this weighted adjacency matrix would give the weights of the possible indirect edges between any two variables. For example, the product $W_{ik}^* W_{kj}^*$ is the possible weight of an indirect path observed from i to j such that the intermediary variable is k and hence, ij^{th} element of $(W^*)^2$ would give the total weight corresponding to all indirect paths from i to j with a single intermediary variable. Similarly, ij^{th} element of $(W^*)^3$ would give the total weight corresponding to all indirect paths from i to j with two intermediary variables and so on. Now, if we construct a graph that captures all possible direct and indirect causal interactions, then its weighted adjacency matrix can be written as follows:

$$W = W^* + (W^*)^2 + (W^*)^3 + \dots + (W^*)^\infty \quad (2.9)$$

W is the weighted adjacency matrix of the entire graph that contains information about both direct and indirect causal interactions present in the system. This matrix W is referred to as the weighted reachability matrix of the system in this article, similar to the reachability matrix defined as the sum of powers of adjacency matrices [47]. W^* is the true weighted adjacency matrix of the system that shows the direct edges alone.

For any matrix H , its geometric series $(\sum_{m=0}^{\infty} H^m)$ converges iff the absolute values of all eigen values of H are less than 1, and it converges to $(I - H)^{-1}$ [49]. Using this property, we can rewrite the equation (2.9) as follows:

$$W = \sum_{m=1}^{\infty} (W^*)^m \approx (I - W^*)^{-1} - I \quad \text{if } \max(|\lambda_{W^*}|) < 1 \quad (2.10)$$

λ_{W^*} represents the eigen values of W^* . Rearranging the above equation, we get

$$W^* = I - (I + W)^{-1} \quad (2.11)$$

The above analytical expression can be used to convert the weighted reachability matrix (W) to the true weighted adjacency matrix (W^*). The graph corresponding to W can be thought of as the graph that we observe from the data of a system as the data would contain information about both direct and indirect causation present in that system. Similarly, the graph corresponding to W^* can be thought of as the desired causal map of the system as it contains only the direct causal information. Thus, we can use the above relation to transform the observed causal map to the desired causal map of the system. However, this equation is valid if and only if $\max(|\lambda_{W^*}|) < 1$. For a system/process that can be represented in the form of a DAG without any self-loops (referred to as DAG systems in the following discussions), this property holds true.

For DAG systems, we can always find a particular order of variables such that its corresponding weighted adjacency matrix is upper-triangular³. Hence, its eigen values are its diagonal elements. As self-loops are not allowed, its diagonal elements are all zero and thus the weighted adjacency matrix is always strictly upper triangular. Hence, for such a system, all eigen values of the true weighted adjacency matrix will always be zero. This particular feature allows us to use the above approximations for DAG systems. A key point to remember is that in reality, we do not know the true weighted adjacency matrix (W^*) and hence, we cannot verify whether this approach is applicable. However, there is another important property of DAG systems or strictly upper triangular matrices. Powers of a strictly upper triangular matrix are also strictly upper triangular which implies that W , which is the sum of all powers of W^* , will be strictly upper triangular. Hence, all eigen values of W will also be zeros. In other words, if the true system can be represented using DAG, the graph that we observe based on data which contain all indirect interactions will also be a DAG. The converse is also true. Hence, it can be concluded that if the observed graph is a DAG, we can reduce it to its true causal map using equation (2.11).

³This property is also applicable to adjacency matrix. As adjacency matrix can be thought of as a special case of weighted adjacency matrix with all weights as 1, the discussion is presented in terms of weighted adjacency matrix.

Observed Graph (based on \hat{W})

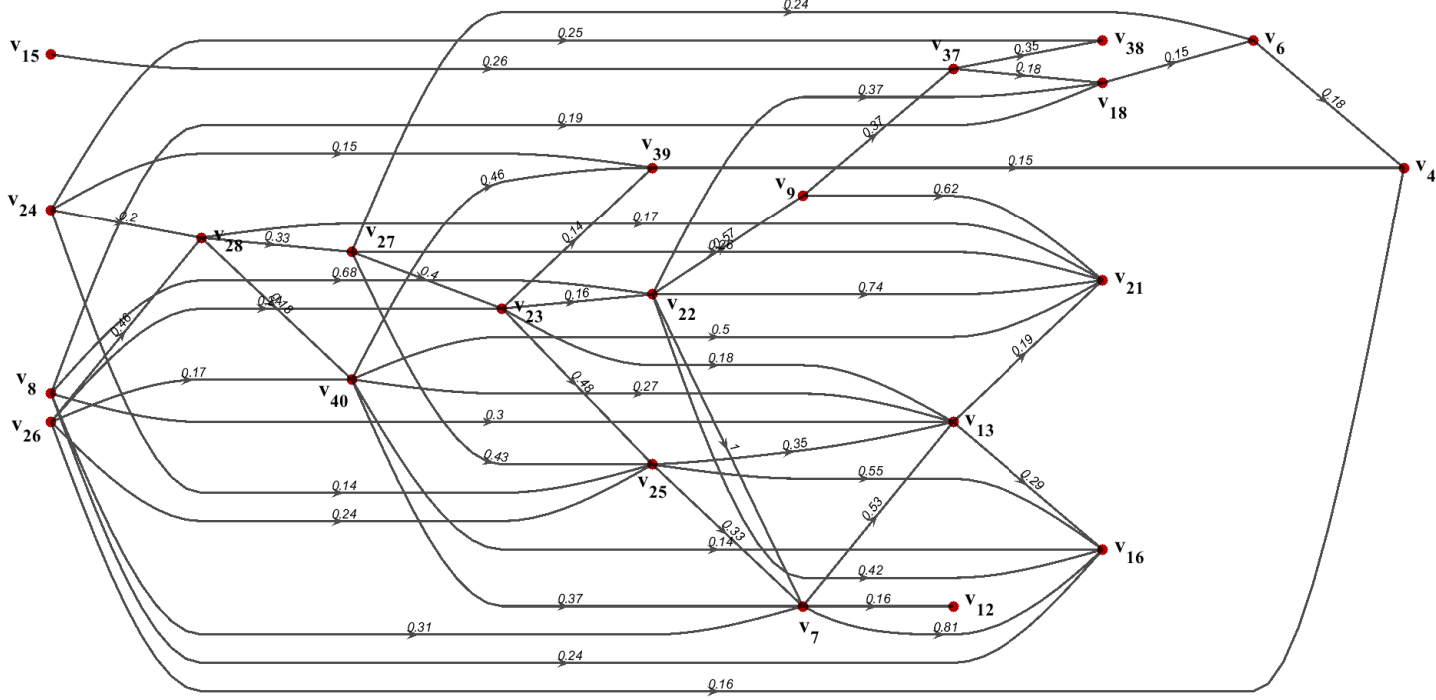


Figure 2.8: Plant-level DAG constructed based on transfer entropy for the Tennessee Eastman benchmark process (Base case) by setting $w_0 = 0.14$ as threshold.

Reduction to obtain DAG As discussed in section 2.3.1, though we chose variables such that actual cyclic effects are insignificant, the plant-level digraph obtained based on transfer entropy contains cycles. These cycles captured could be due to the noise present in the data or due to the presence of indirect edges (resulting from interdependencies between variables). This speculation is supported by the fact that among the edges that form a cycle, at least one of the edges has a very small weight. In order to obtain a DAG from this graph, the best way is to increase the threshold value w_0 . By choosing a suitable w_0 (for equation (2.8)) such that all eigen values of \hat{W} are zeros, we can ensure that \hat{W} is a DAG.

This approach was tested for the Tennessee Eastman case study discussed in section 2.3.1. This analysis showed that by setting $w_0 = 0.14$, we obtain a DAG. The corresponding

causal map obtained is shown in Figure 2.8 and it can be seen that there are no cycles in this graph. The total number of edges captured in this graph is 51.

Weakening of indirect edges As we have a DAG for the process based on the variables chosen, we can use the transformation expressed in equation (2.11) to obtain the true graph of the process. However, it is important to note that this equation is obtained based on the assumption that the observed weighted adjacency matrix (W) incorporates all possible indirect interactions between variables and is equivalent to the weighted reachability matrix. This may not be always true. Hence, in reality, the observed weighted adjacency matrix based on data (\hat{W}) may capture only a subset of indirect causal interactions. However, it can be observed that though this transformation is not capable of completely removing the indirect edges, it certainly weakens their weight. Hence, this expression can still serve as an approximate solution to direct causality detection problem. The weighted adjacency matrix obtained after transformation will have elements lesser than the threshold w_0 and those have to be removed. By thresholding the weights using w_0 , the final plant-level DAG for the system can be represented as ,

$$\hat{W}_{\text{DAG}} = \begin{cases} (I - (I + \hat{W})^{-1}) & \text{if } (I - (I + \hat{W})^{-1}) \geq w_0 \\ 0 & \text{if } (I - (I + \hat{W})^{-1}) < w_0 \end{cases} \quad (2.12)$$

$$\hat{W}_{\text{DAG}} = \frac{\hat{W}_{\text{DAG}}}{\max(\hat{W}_{\text{DAG}})} \quad (2.13)$$

A simple case study is given in Figure 2.9. For this system, a DAG was obtained for a zero threshold ($w_0 = 0$). The figure shows how the transformation using equations (2.12, 2.13) helps to weaken the weight corresponding to indirect edges. Weighted adjacency matrix calculated based on the data simulated using MATLAB Simulink showed an indirect edge from v_1 to v_3 and interestingly, that indirect edge had the highest significance value and hence $\hat{W}_{13} = 1$. However, after transformation using equations (2.12, 2.13), its weight was

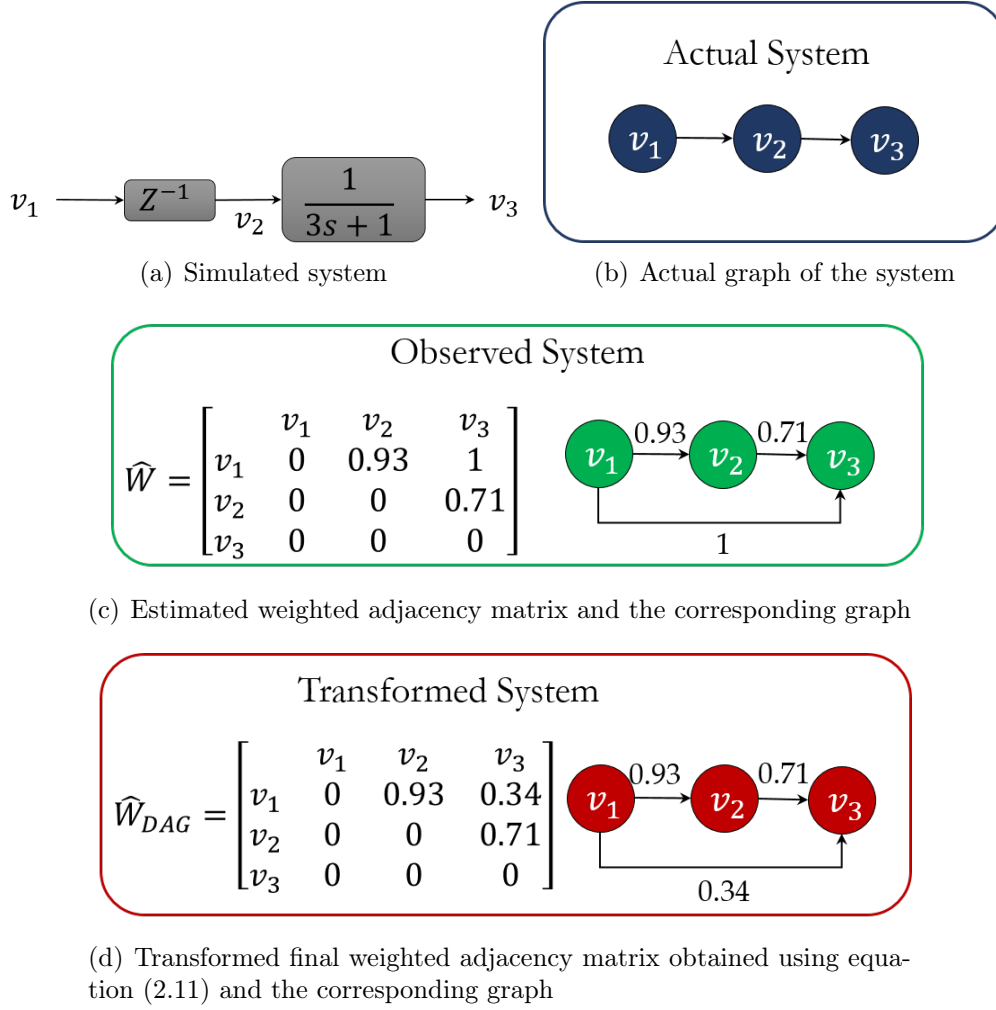


Figure 2.9: A simple case study to show the direct causality detection algorithm. \hat{W} shows a weight of 1 for the indirect edge, whereas its weight was reduced to 0.34 after transformation.

reduced to 0.34 ($\hat{W}_{DAG_{13}} = 0.34$). This shows the importance of this approach in weakening the weights of indirect edges.

For the Tennessee Eastman case study discussed in section 2.2, the same procedure was adopted to calculate \hat{W}_{DAG} . The graph obtained based on \hat{W}_{DAG} is provided in Figure 2.10. It can be seen that, there are only 44 edges in the graph unlike 51 in the graph based on \hat{W} (Figure 2.8). The weights corresponding to the remaining 7 edges were weakened during transformation and thus were removed in the final graph. For example, initial observation shows a causal interaction from reactor level (v_8) to stripper temperature (v_{18}) which is

Transformed graph (based on \hat{W}_{DAG})

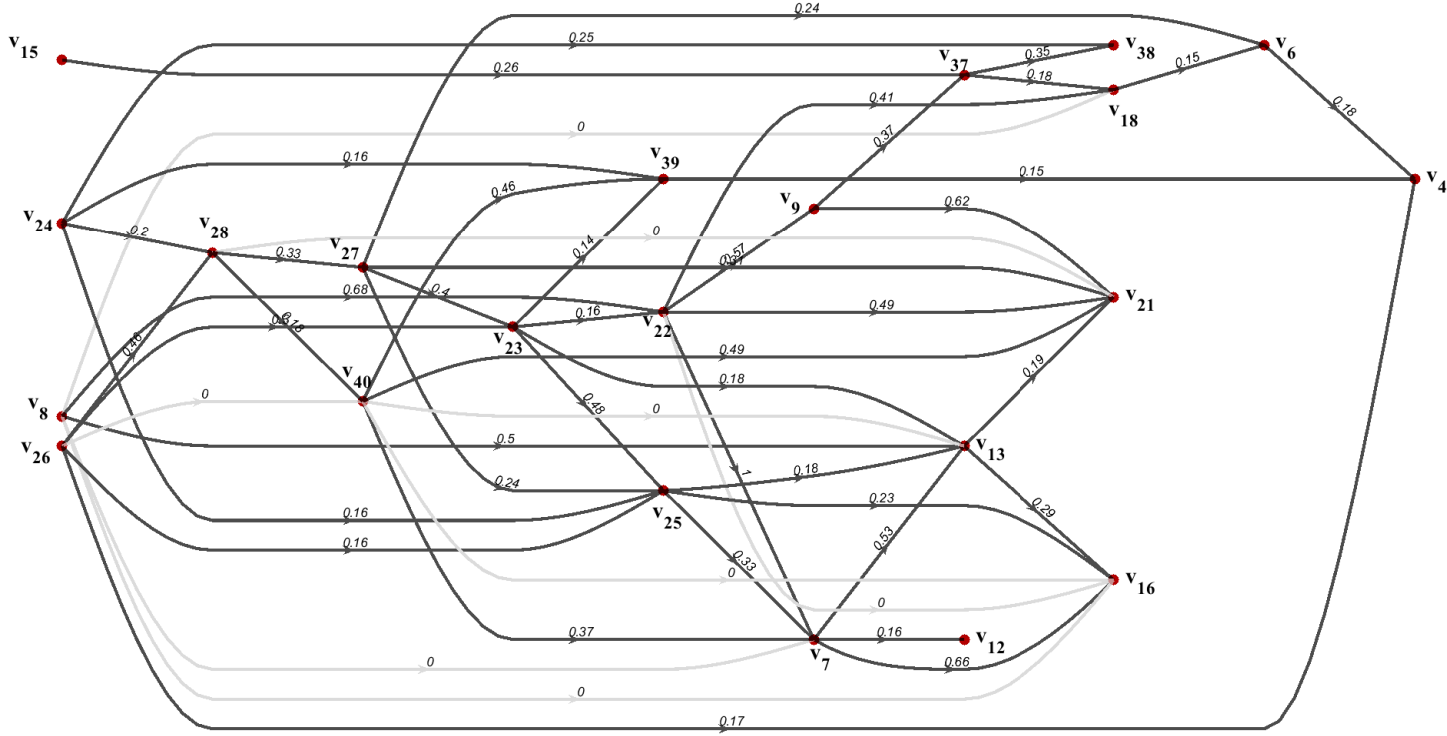


Figure 2.10: Final plant-level DAG constructed based on transfer entropy for the Tennessee Eastman benchmark process (base case) after transformation. $w_0 = 0.14$ is used as threshold. The gray arrows are the once which were nullified in comparison to the originally obtained graph

removed after transformation. As condenser cooling water outlet flow rate is manipulated based on reactor level, reactor level can be considered the cause of both cooling water outlet temperature (v_{22}) and product separator temperature (v_{11}). v_{11} is in turn a cause for stripper temperature (v_{18}). As product separator temperature is not considered for analysis, it is acceptable to say that v_{22} causes v_{18} . The initial causal observation from v_8 to v_{18} is hence a spurious connection due to the indirect path between these through an intermediary variable v_{22} . Therefore, it can be seen that the transformation algorithm correctly identifies the indirect edge. By comparing the final plant-level graph (Figure 2.10) with that of the process

flowsheet given in Figure 2.3, it can be seen that the edges derived based on the proposed algorithm correspond to the actual causal interactions present in the system, though a few may be indirect effects.

The most important root nodes obtained from this analysis on the Tennessee Eastman case study are the stripper level (v_{15}), reactor level (v_8), composition of B (v_{24}) and D (v_{26}) in stream 6. The strongest connection in the obtained graph is from v_{22} to v_7 , corresponding to the separator cooling water outlet temperature to the reactor pressure. Note that the separator cooling water outlet temperature is related directly to the separator temperature, which will affect the reactor pressure as seen from Figure 2.3 (with the help of compressor). Apart from this connection, connection from v_8 to v_{22} (reactor level to separator cooling water outlet temperature), v_7 to v_{16} (reactor pressure to stripper pressure), and v_9 to v_{21} (reactor temperature to reactor cooling water outlet temperature) are some of the other prominent ones. The following nodes (arranged in the order of significance) are correctly identified as sink nodes: stripper pressure (v_{16}), reactor cooling water outlet temperature (v_{21}), composition of E in stream 11 (v_{38}) and composition of H in stream 11 (v_{41}). Though the algorithm also identifies product separator level (v_{12}) as one of the sink nodes, the corresponding edge has a very small weight associated with it. From these observations, it can be seen that the proposed algorithm for plant-level digraph construction indeed captures the overall causal relations present in the process.

2.3.2 Tier 2: Subsystem-level graph with possible cycles

All subsystems containing some sort of cycles, like recycle streams or feedbacks, are analyzed separately to construct a subsystem-wise causal map. The number of subsystem-level graphs and the variables in each such graph have to be decided based on prior knowledge about the system in the form of process flowsheet. The overall procedure of selection of variables and units are explained below with the help of the Tennessee Eastman case study provided in Figure 2.3.

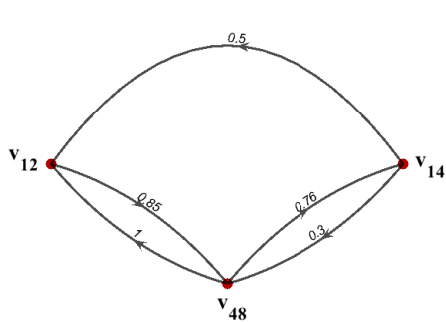
Table 2.3: List of subsystem-level causal maps

Number	Description	Variables
1	Controller for concentration of A and C in feed	$v_1, v_4, v_{23}, v_{25}, v_{44}, v_{45}$
2	Controller for concentration of G in product	$v_2, v_3, v_{40}, v_{42}, v_{43}$
3	Controller for reactor temperature	v_9, v_{51}
4	Controller for reactor pressure	v_7, v_{10}, v_{47}
5	Controller for reactor level	v_8, v_{11}, v_{52}
6	Controller for separator level	v_{12}, v_{14}, v_{48}
7	Controller for stripper level/product flow rate	v_{15}, v_{17}, v_{49}
8	Controller for compressor	v_5, v_{20}, v_{46}
9	Feed as a mixture of various streams	v_1, v_2, v_3, v_5, v_6

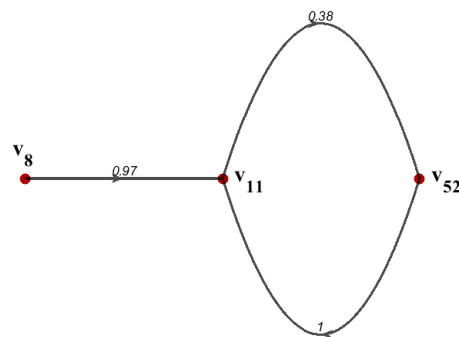
Every control-loop can be associated with a corresponding graph and can be analyzed separately. For example, in the Tennessee Eastman case study, there are 12 manipulated variables which are associated with 8 different control loops (Note: multiple levels in a multi-level controller are considered together). By analyzing the set point, error, controller output, valve output (manipulated variable) and the controlled variable for each control loop, a graph is constructed for that control loop. As there is a feedback in case of a control loop, the associated true graph will have cycles. The causal graph is hence, created using equation 2.7 (and not equation 2.8 as it is applicable only for systems without cycles). It is to be noted that data-driven approaches including transfer entropy will show interactions between all variables. However, the intensity of each interaction would vary.

In addition to control loops, we should also construct graphs for systems with recycle streams. For example, streams 1, 2, 3, 6 and 8 can be used to make a causal map which will capture the mixing of various streams to give stream 6 and the effect of recycle stream 8 on the reactor feed.

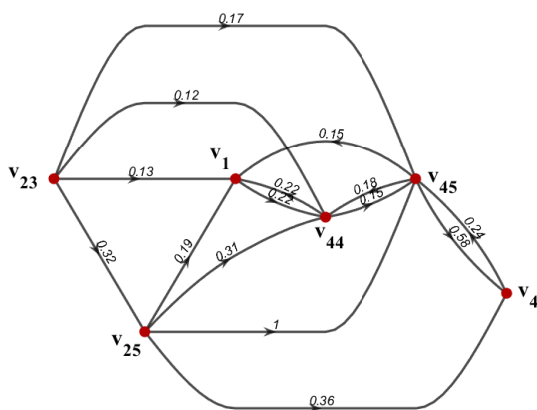
Overall, we will have 9 subsystem-level causal maps for this case study. Table 2.3 shows the list of all subsystem-level causal maps and Figure 2.11 shows a few of these causal maps generated using the data collected for base case scenario. Note that these graphs contain cyclic components and bidirectional edges as they are intended to capture various cyclic interactions present in the system. Hence, the transformation discussed in the previous section cannot be applied for these types of graphs. Figure 2.11(a) shows the control loop



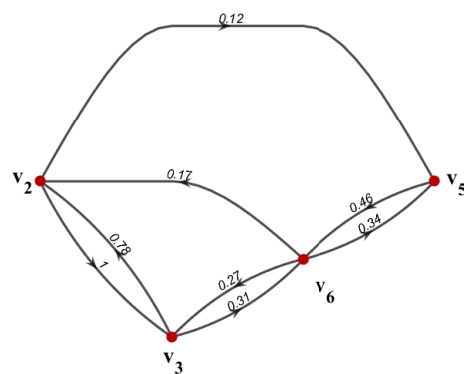
(a) Controller for product separator level, Loop 10



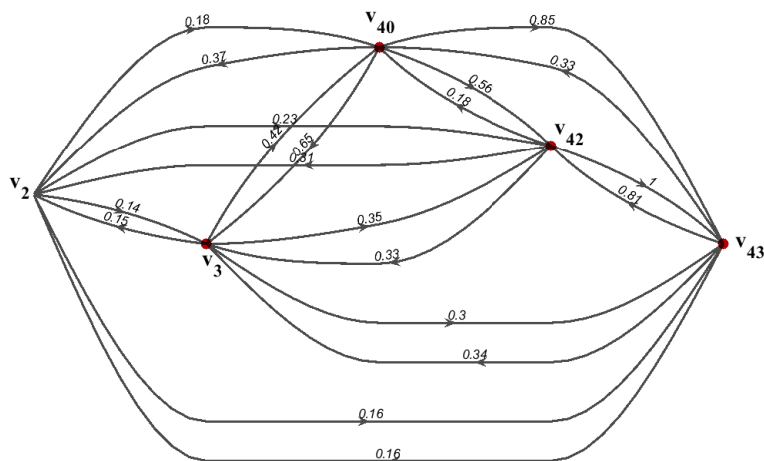
(b) Controller for reactor level, Loop 11



(c) Controller for concentration of A and C in feed



(d) Feed as a mixture of various streams



(e) Controller for concentration of G in product, Loop 13

Figure 2.11: A few examples of subsystem-level causal maps for the Tennessee Eastman case study.

for the product separator level (LC/10 and RC/6 in Figure 2.3) in which it is shown that the major effect is seen from separator pot liquid flow to product separator level, which is expected from a first principles standpoint. Also, it shows no connection from v_{12} to v_{14} as their causal effect is through the intermediate manipulated variable v_{48} . In Figure 2.11(b), causal map related to reactor level controller is shown. The most prominent effects are from condenser cooling water flow to product separator temperature, and reactor level to product separator temperature. It also shows that the lower level manipulated variable v_{52} is not affected directly by the controlled variable of higher level controller v_8 , instead the effect is through the intermediate variable v_{11} . These effects can be inferred from Figure 2.3.

2.4 Major Results

Major contributions of this work is the hierarchical approach that decouples the cyclic and acyclic interactions present in the complex system, and the transformation using the ideas of reachability and adjacency matrices. Hierarchical approach requires us to manually choose the variables based on the information from flowsheets and this step is performed off-line. The transformation involves simple matrix inverse and multiplication operations which are computationally very fast. So, the computational time essentially depends on the measure of causality used for analysis. Causality metric can be chosen based the system of interest.

Transfer entropy is used as the metric for causality as it is a reasonable metric for non-linear systems. Transfer entropy is known to be computationally expensive. Additionally, we use surrogate data for hypothesis testing. As it is required to estimate transfer entropies for each surrogate pair, the computational time also depends on the number of surrogate pairs chosen. It took approximately 1.7 hours (in a 16GB RAM Intel Core i7-2600 CPU) to generate the complete plant-level causal map when 20 surrogate pairs were used for hypothesis testing. Instead, if we were to use cross-correlation for quantifying causality, it takes approximately 1 minute to generate a plant-level causal map on the same computer. So,

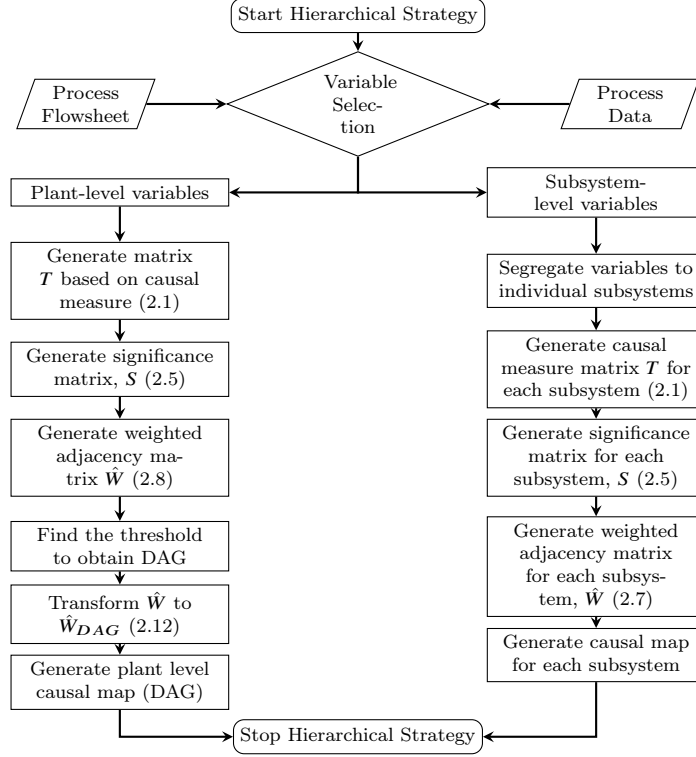


Figure 2.12: Flowchart of the proposed hierarchical approach.

there will be a trade-off between accuracy and computational time. Based on the application, a reasonable metric can be chosen. The proposed approach of hierarchical analysis and transformation is generalizable to any causal metric (Figure 2.12).

Robustness of this approach again depends on the chosen causality metric. For transfer entropy-based causality metric discussed in the manuscript, the edge weights in the causal map are based on the significance values obtained during hypothesis testing. Hence, the robustness of the transfer entropy metric is heavily dependent on the number of surrogate pairs chosen. The results will be robust if a reasonably large number of surrogate pairs are generated for hypothesis testing.

Chapter 3: Neural Networks for Classification

Deep neural networks have evolved into a powerful tool applicable for a wide range of problems. However, a clear understanding of their internal mechanism has not been developed satisfactorily yet. Factors such as the architecture, number of hidden layers and neurons, and activation function are largely determined in a guess-and-test manner that is reminiscent of alchemy more than of chemistry. In this chapter, we attempt to address these concerns systematically using carefully chosen model systems to gain insights for classification problems. We show how wider networks result in several simple patterns identified on the input space, while deeper networks result in more complex patterns. We show also the transformation of input space by each layer and identify the origin of techniques such as transfer learning, weight normalization and early stopping. This chapter is an initial step towards a systematic approach to uncover key hidden properties that can be exploited to improve the performance and understanding of deep neural networks.

In recent years, deep neural networks have witnessed great success and widespread acceptance for solving many challenging problems in different fields. As observed by Venkatasubramanian [1], this explosive growth is due to three key developments: availability of large amounts of data (i.e., "big data"), availability of powerful and cheap hardware, and advances in user-friendly software environments, all of which can be traced directly to the unexpected success of Moore's Law over the last fifty years. While many ideas such as the back propagation algorithm, convolutional neural nets, recurrent neural nets, Bayesian learning, reinforcement learning, etc., have been around for a couple of decades or more, and therefore are not that "new", what is "new", however, is the ease with which these, and other such techniques, can now be applied in a variety of domains due to the three factors mentioned above.

This ease of application of powerful tools has the potential to cause their misuse and abuse, if one is not careful. This concern is similar to what we have witnessed over the years with statistics [11]. In this regard, the current state of machine learning is akin to that of alchemy before chemistry and chemical engineering, involving a lot of guess-and-test trials, as noted by Jordan [12] and Hutson [13].

Important questions such as what kind of neural network to use, how many hidden layers, how many neurons in each layer, what kind of activation or squashing function to use, how to initialize the weights, and on and on, are largely determined by an Edisonian guess-and-test fashion that reminds one of alchemy more than of chemistry, as noted. A systematic approach to answering these design questions is largely absent in many applications, and such decisions are highly dependent on the expertise (or the lack thereof) of the programmer. Despite the seeming practical success of deep neural nets, we find this state of affairs deeply unsatisfactory for at least three reasons. First, it reveals our lack of fundamental understanding of the theory of deep neural networks. Second, this makes the use of neural networks in some engineering applications worrisome due to their lack of generalizability and to their potential exposure to adversarial attacks. Such drawbacks are perhaps not very serious in recognizing cats vs dogs, or in recommendation systems (e.g., Yelp, Rotten Tomatoes, etc.), but can be quite important in applications such as fault diagnosis, process safety analysis, where the cost of a mistake could be quite high. Third, there is no causal explanation provided which is needed in many engineering applications.

This chapter is written in a tutorial-like manner to highlight these concerns, particularly for the newcomers in machine learning, and to explore and understand the hidden internal representations of deep neural networks to gain some insights for classification problems (the second part of this series explores function approximation problems, to be published shortly). We have chosen deliberately simple examples to reveal key insights about the structure and behavior of deep neural nets, such as the role of depth vs breadth of layers, the different activation or squashing functions and so on. The kinds of examples we have

chosen are to be viewed like the "particle in a box" or "simple harmonic oscillator" case studies in quantum mechanics, or like the "ideal gas" system in statistical mechanics. As we know, despite their manifest simplicity, such models have been extraordinarily helpful to understand and appreciate the critical conceptual issues and their interrelationships. They often reveal subtle patterns, and in the case of deep neural networks literally hidden internal representations, which are often hard to perceive in more complex examples.

There have been attempts in the past to explain the functioning of a neural network, using visualization techniques [50], rule extraction [51], randomization approaches to give insights into the importance of nodes in a neural network [52], and recently the information bottleneck approach which seems to give the most insight into the functioning of a neural network [53, 54, 55].

However, from a network design point of view, there are several questions facing a user that remain largely unanswered, rendering the task of designing a network for a certain objective a highly trial-and-error exercise. In addition, there is a general tendency to build deep networks with complicated units such as convolutional filters, recurrent units and inception modules that result in better performance without a clear understanding of the trade-offs involved. It is important to know the minimal neural net architecture for a given problem. Conventionally, an architecture is chosen based on the complexity of the data distribution, number of features, and so on. There is a belief that the deeper the neural network, the better it works - why that is the case is often left as a mystery.

The lack of a clear understanding of the internal mechanism of neural networks partly stems from the intractability of the operations performed by the network. A neural network consists of several nodes arranged in layers that can be connected in different layouts, resulting in intractable operations of the overall network. In this chapter, we adopt a systematic approach and make an attempt to understand the operation of the individual components, their arrangements, and finally an entire network. In that sense, our cases represent simple yet rich exemplars (along the lines of the simple harmonic oscillator, particle in a box, and

ideal gas molecules, as noted) to gain a deeper understanding. We proceed from an understanding of the individual elements to that of the interactions between them in order to understand the operation of intermediate layers, and then of the complete neural network.

We hope our findings will be of pedagogical value to the new practitioners in the field. Our style of investigation should be of particular interest to chemical engineers who have always prided themselves, historically, as first-principles-based modelers, for whom any "black box" model would be an anathema. This chapter is an attempt to inject some transparency into the box, making it more like a "gray box".

In this chapter, we attempt to shed light into the inner workings of the information transformations in a deep neural network. We test simple examples to understand critical issues with the training process. We examine loss function landscapes and show that degeneracy plays a critical role in the parameter space. This in turn results in a degeneracy of observations. Finally, we show what features are estimated by a neural network during the training process for different activation functions. We study three examples which have nonlinear decision boundaries to elucidate this. The generalization of the identified feature space is also studied for a chemical engineering fault diagnosis problem.

3.1 Mathematical Background

3.1.1 Problem Formulation

The dataset in classification tasks comprises (N) examples of input data and their corresponding target classes, represented here as $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N$ where $\mathbf{x}^{(i)} \in \mathbb{R}^m$ represents the i^{th} input data and $\mathbf{y}^{(i)}$ is a K -dimensional one-hot vector representing the target vector corresponding to $\mathbf{x}^{(i)}$. The problem of classification is mathematically formulated as an optimization problem wherein the objective (or loss) function is the cross-entropy between the

target vector $\mathbf{y}^{(i)}$ and the predicted outputs $\hat{\mathbf{y}}^{(i)}$, defined as:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^K y_j^{(i)} \log \hat{y}_j^{(i)} \quad (3.1)$$

The target vector as well as predictions in Equation (3.1) represent the probability mass functions (of the target and predicted classes respectively) so that the quantity in Equation (3.1) represents the deviation of the predicted outputs (probabilities) from the target, averaged over all examples.

In the above context, a neural network is then a model \mathcal{N} with a predefined architecture \mathcal{A} and a set of parameters Θ that expresses the output as a function of the inputs as:

$$\hat{\mathbf{y}}^{(i)} = \mathcal{N}(\mathbf{x}^{(i)}; \mathcal{A}, \Theta) \quad (3.2)$$

In this article, we consider the architecture \mathcal{A} to include the design choices made by the user and differentiate them from the parameters Θ that are tuned during training. The architecture \mathcal{A} therefore includes:

1. the organisation of layers, eg., fully connected, convolutional, recurrent
2. the activation function: $g(\cdot)$, eg., logistic function, hyperbolic tangent, rectified linear unit
3. the number of layers L (values ranging from < 10 to $\simeq 150$ in the literature)
4. the number of nodes in each layer n_l , $l = 1, 2, 3, \dots, L$ (values ranging from ten to few hundreds in the literature)

On the other hand, the parameters Θ include the weights $\mathbf{W}^{[l]}$ and biases $\mathbf{b}^{[l]}$ of each layer, $l = 1, 2, 3, \dots, L$. It is to be noted that once the architecture \mathcal{A} is defined, the dimensionality of the weights and biases and hence, dimensionality of Θ is fixed.

3.1.2 Classification with Neural Networks

The network \mathcal{N} is required to identify (*learn*) the boundaries that can be used to differentiate data belonging to one class from others. These boundaries, in the simplest case, form planes (referred to as *separating hyperplanes*) that divide the input space into multiple regions allowing one to assign different class labels to different regions. However, in several applications these boundaries are extremely curved surfaces, referred to as *separating hypersurfaces*. The nature of the separating boundary results in a linear or nonlinear classification problem, and determines the complexity of a model required to reliably identify the boundaries. It must be noted that in some applications such as identifying lion cubs from cats, it is not possible to manually identify distinct hypersurfaces in the input space, and it becomes necessary to extract certain *features* relevant to the classification problem. Therefore, achieving classification with neural networks, which entails identifying appropriate separating boundaries from the data typically involves *feature extraction* followed by *classification*. While the hidden layers are used to extract relevant features from the data, the final layer is used to identify the separating surfaces necessary for classification.

Let us consider a multi-layer network \mathcal{N} with L layers and the l^{th} layer consisting of n_l nodes and an activation function $g^{[l]}(\cdot)$. Let $\mathbf{z}^{[l]}$, $\mathbf{W}^{[l]}$, $\mathbf{b}^{[l]}$, $g^{[l]}(\cdot)$ and $\mathbf{a}^{[l]}$ represent the input, weights, biases, activation function and output of the l^{th} layer respectively. The output $\hat{\mathbf{y}}$ of the network can then be obtained through a progression of layer-wise processing of the input \mathbf{x} as:

$$\begin{aligned}\mathbf{a}^{[1]} &= g^{[1]}(\mathbf{z}^{[1]}) = g^{[1]}(\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}) \\ \mathbf{a}^{[l]} &= g^{[l]}(\mathbf{z}^{[l]}) = g^{[l]}(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}) \quad \forall l \in \{2, \dots, L-1\} \\ \hat{\mathbf{y}} &= g^{[L]}(\mathbf{W}^{[L]}\mathbf{a}^{[L-1]} + \mathbf{b}^{[L]})\end{aligned}\tag{3.3}$$

The output $\hat{\mathbf{y}}$ is therefore a function of the inputs \mathbf{x} , the parameters $\mathbf{W}^{[l]}, \mathbf{b}^{[l]}$, $l = 1, 2, 3, \dots, L$ and the activation function $g^{[l]}(\cdot)$, $l = 1, 2, 3, \dots, L$. It must be noted here that

while the expressive power of a neural network lies in the architecture chosen for a particular task, its actual performance depends on the architecture as well as the values of the parameters. The host of algorithms developed for training neural networks attempt to tune the parameters Θ of the network \mathcal{N} so as to minimize the loss. In the following, we discuss these factors and highlight the different scenarios considered in this work.

The choice of the the number of nodes in a layer (width), number of hidden layers (depth), type of connection (fully connected, convolutional, recurrent) and the activation function (denoted as $g(\cdot)$) used in each layer together determine the expressive power of a neural network. In this work, we consider multiple combinations of width and depth for fully connected networks to identify the influence of these factors on the performance of the network. In addition, we consider the logistic function σ , hyperbolic tangent function *tanh*, Gaussian function G and the rectified linear unit *ReLU* as different activation functions used in the hidden layers in the networks. Table 3.1 lists the definitions and ranges of the different activation functions used in this study. The different activation functions are also shown in Figure 3.1.

Table 3.1: Activation functions and their ranges

Activation Function (g)	Expression	Range
Logistic Function (σ)	$\frac{1}{1 + \exp(-z)}$	$[0, 1]$
Hyperbolic tangent (<i>tanh</i>)	$\frac{\exp(z) + \exp(-z)}{\exp(z) - \exp(-z)}$	$[-1, 1]$
Gaussian Function (G)	$\exp(-z^2)$	$[0, 1]$
Rectified Linear Unit (<i>ReLU</i>)	$\max(0, z)$	\mathbb{R}^+

In addition to the above, the final layer of the network consists of the *softmax* activation function, which is used to produce the probability of each class for the input, and is defined for a layer with K nodes (corresponding to a K - class problem) as follows:

$$P(y = j) \triangleq \hat{y}_j = \frac{\exp(z_j)}{\sum_{k=1}^K \exp(z_k)} \quad \forall j \in \{1, \dots, K\} \quad (3.4)$$

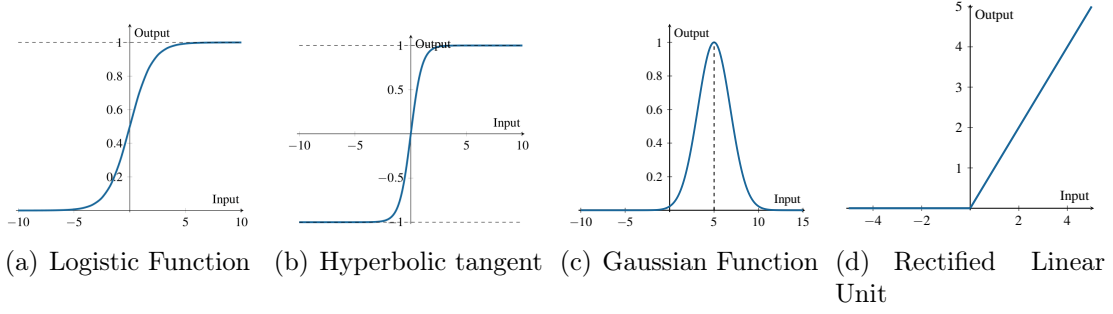


Figure 3.1: Activation functions used in this study

In the following, we present a systematic approach to understanding the internal mechanism of a neural network. We begin with a simple network that is the equivalent of a single-cellular organism, or a particle in a box and that lends itself to a dissection-like study providing insights about larger networks with complicated architectures. We then investigate the impact of different components of the architecture \mathcal{A} listed earlier on the performance of a neural network.

3.2 Peeking Under the Hood of a Deep Neural Network

A neural network is made up of multiple neurons arranged in a layer that are connected to neurons in other layers. Each node in the l^{th} hidden layer of a network activates a portion of the space spanned by the nodes of $l - 1^{\text{th}}$ layer, according to the parameters $\mathbf{W}^{[l]}, \mathbf{b}^{[l]}$ and the activation function $g^{[l]}(\cdot)$. As discussed earlier, a network performs *feature extraction* from the input data with the hidden layers, which are then used for performing *classification* at the final layer, which are analysed in the same order in the following.

3.2.1 Feature Extraction: Node-specific Selective Activation of the Input Space

Let us consider a tiny neural network that consists of two input nodes and one output node as shown in Figure 3.2. The output activation of the network can be expressed as:

$$a = g(w_1x_1 + w_2x_2 + b)$$

For this network, let us define a node-specific *characteristic equation* that determines the activation of the input space as:

$$w_1x_1 + w_2x_2 + b = 0 \tag{3.5}$$

The above equation describes a line in the 2 dimensional input space that acts as the reference for selectively activating different regions of the space.

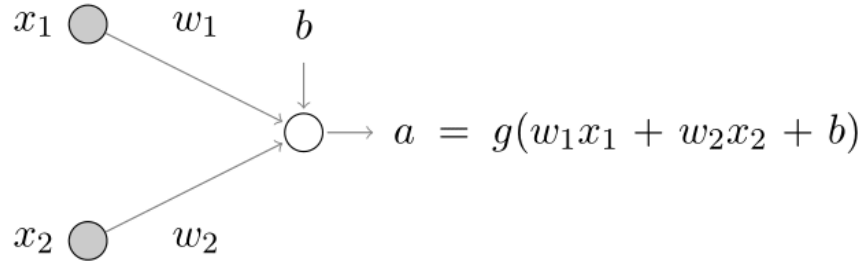


Figure 3.2: Architecture for 2 input neurons and 1 activated neuron

Figure 3.3 shows the output of the network for different activation functions $g(\cdot)$ used in the output node. In each plot, the set of points satisfying the characteristic equation (with $w_1 = -1$, $w_2 = 1$ and $b = 0$) is shown as a red line, while the value of the output is shown as a gradient of colour with yellow representing the highest value and deep blue representing the lowest value. It is noteworthy that each point (X_1, X_2) in the input space is activated according to the scaled, rotated and translated value $d(X_1, X_2) = w_1X_1 + w_2X_2 + b$, which is also equal to the scaled directional distance of the point (X_1, X_2) from the line segment representing the characteristic equation, scaled by the norm of $\mathbf{w} = [w_1, w_2]$. Henceforth d

shall be referred to as the distance.

Different types of activation functions however result in different segments of the input space being activated as shown in Figure 3.3. It can be observed from Figure 3.3 that while σ results in the line $w_1x_1 + w_2x_2 + b = 0$ being assigned a value of 0.5, \tanh suppresses the same to zero, activating other points according to their distance from the line. On the other hand, $ReLU$ suppresses all inputs that satisfy $w_1x_1 + w_2x_2 + b \leq 0$ while activating others by an amount equal to their magnitude. Finally, the Gaussian activation results in the line $w_1x_1 + w_2x_2 + b = 0$ being activated with the highest magnitude, with the activation decaying as a function of the distance of the data points from the line.

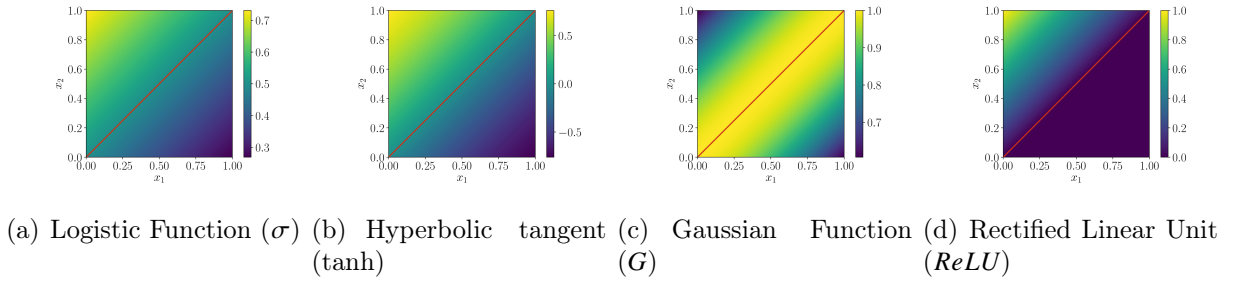


Figure 3.3: Activation of the network shown in Figure 3.2 for different regions of the input space with different activation functions. The red line shows the set of points satisfying $w_1x_1 + w_2x_2 + b = 0$ ($w_1 = -1, w_2 = 1, b = 0$).

As mentioned earlier, in addition to the form of the activation function, the magnitude of activation of an input data point (X_1, X_2) also depends on its distance from the line representing the characteristic equation, equal to $w_1X_1 + w_2X_2 + b$ which is tuned at the time of training. This has two implications on the training of neural networks. *First*, the parameters w_1 , w_2 and b can be tuned over a continuum of values resulting in different lines in the input space and thus different levels of activation. *Second*, it must be noted that the set of points that satisfy $w_1x_1 + w_2x_2 + b = 0$ also satisfy the equation $\alpha w_1x_1 + \alpha w_2x_2 + \alpha b = 0$ for all non-zero values of α . As a result, the magnitude of activation of any given input data point can also be tuned over a continuum of values by adjusting the value of α . Furthermore, the values assigned to data points on different sides of the line can be interchanged by accommodating

negative values of α . Figure 3.4 shows the the output of the network for different activation functions $g(\cdot)$ with set of points satisfying the characteristic equation (with $w_1 = -3$, $w_2 = 3$ and $b = 0$) is shown as a red line and a similar gradient of colour representing different levels of activation. A comparison of Figures 3.3 and 3.4 reveals that the latter results in a much greater gradient and/or magnitude of activation resulting from the larger value of $\|\mathbf{w}\|$ for the network. This demonstrates the impact of α on the activation levels for each node, and the extent of nonlinearity achieved with the same activation function.

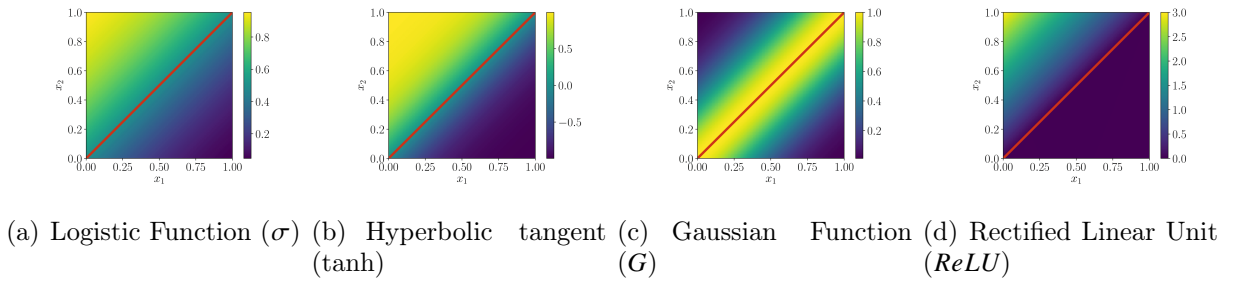


Figure 3.4: Activation of the network shown in Figure 3.2 for different regions of the input space with different activation functions. The red line shows the set of points satisfying $w_1x_1 + w_2x_2 + b = 0$ ($w_1 = -3, w_2 = 3, b = 0$)

We discussed the behaviour of a single node in a neural network as a function of the parameters \mathbf{w} , b and the activation function $g(\cdot)$ and identified two different methods of adjusting the parameters to selectively extract features from the input space. In the following, we add more resources to the tiny network and study the impact of depth and width of the resulting network.

3.2.2 Wider vs Deeper Networks: Complexity of Features

Let us add another neuron at the output layer of the tiny network and combine the outputs of the nodes to generate the final output of the new network, as shown in Figure 3.5. The output of this network can now be expressed as:

$$a = g(a_1 + a_2) = g(g(w_1x_1 + w_2x_2 + b) + g(w_3x_1 + w_4x_2 + c))$$

It is now possible to study the activation pattern in each of the three nodes, as shown in

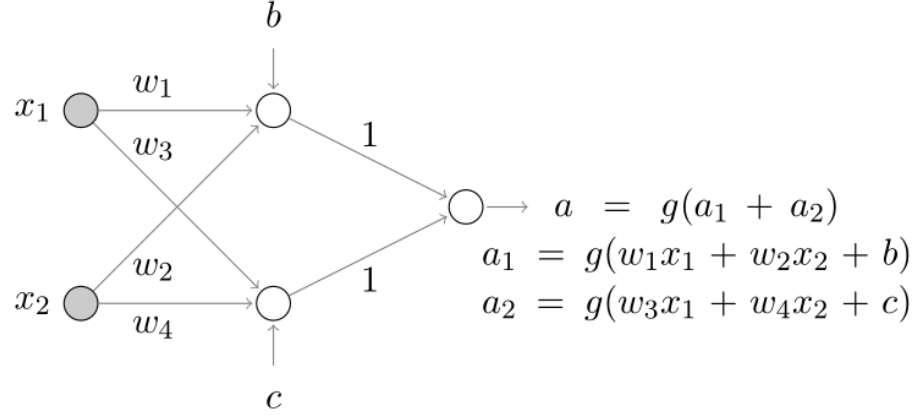


Figure 3.5: Architecture for 2 input neurons and 1 activated neuron, with 1 hidden layer

Figure 3.6 for different activation functions. It can be observed from Figure 3.6 that the two neurons in the hidden layer activate different segments of the input space according to their characteristic equations: $w_1x_1 + w_2x_2 + b = 0$ and $w_3x_1 + w_4x_2 + c = 0$. These activations are then combined to produce the final output that activates different patches in the input space. It must be noted here that while the nodes in the first layer selectively activated the input space according to a characteristic equation representing a line, the node in the final layer activates the input space according to a characteristic equation representing a curve, which is obtained through combination and composition of the different activations.

It can therefore be observed that with the addition of more number of nodes to the tiny network, it is imparted with the ability to generate more number of selective activations, which are *simple* and characterized by a *straight line* in the input space. However, with the addition of more layers to the network, the network can create *complex* selective activations that are characterized by *arbitrary curves* in the input space. These straight lines and curves are tuned at the time of training such that each node in each layer selectively extract a portion of the input space that is relevant to the classification task.

A careful examination of Figures 3.4 and 3.6 reveals that in a network \mathcal{N} with L layers, the nodes in the first layer will always result in activations representing straight lines in the

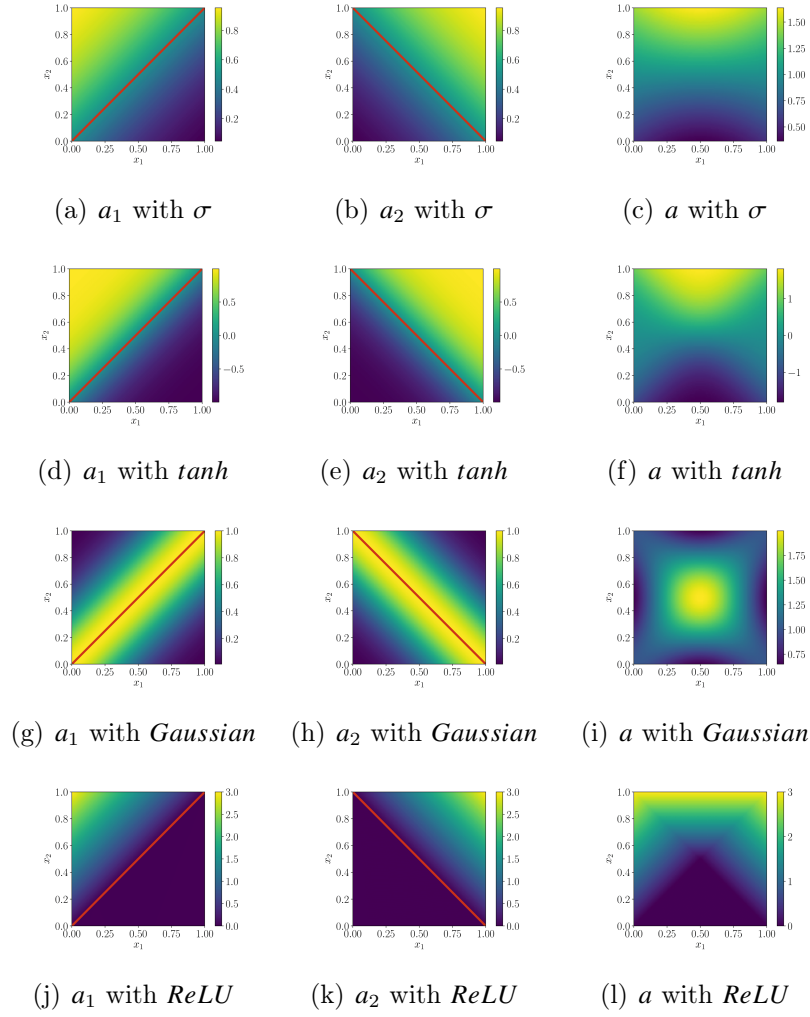


Figure 3.6: Input space activation for each node in the network architecture as shown in Figure 3.5. The values of the weights and biases are chosen to be $w_1, w_2, b = -3, 3, 0$, $w_3, w_4, c = 3, 3, -3$

input space, and as one builds a deeper network, these lines are transformed into curved geometric objects. In other words, the characteristic equation of nodes in the first layer represent straight lines, which become progressively nonlinear as one goes deep into the network - irrespective of the task performed by the network. Now, if two networks are trained to perform different but related tasks, it is likely for the geometric objects identified by the networks to have similar descriptions. As a result of this generic increase in the complexity of the characteristic equation and dependence of the characteristic equation on the final task, it is possible to make use of pre-trained networks as initializations and to train new networks for achieving related tasks with less computational time, a process referred to as *transfer learning* in the literature [56, 57, 58].

For example, consider two networks \mathcal{N}_1 and \mathcal{N}_2 that are trained to achieve different, but related classification tasks, eg., to differentiate cats from other animals (\mathcal{N}_1) and differentiate cats from dogs (\mathcal{N}_2). Both these networks, at the time of training, learn a set of linear and curved geometric objects in the input space that can be used to achieve their corresponding classification tasks. However, since both the networks attempt to distinguish cats from other objects (all other animals for \mathcal{N}_1 and dogs for \mathcal{N}_2), it stands to reason that they identify characteristics explicit to cats in relation to the other objects. As a result, several of the geometric objects identified by \mathcal{N}_1 (to distinguish cats from other animals) are likely to be similar (if not same) to those identified by \mathcal{N}_2 (to distinguish cats from dogs). A pre-trained network that distinguishes cats from other animals and has already identified features relevant to cats can therefore act as a better initialization for a network that is needed to distinguish cats from dogs, resulting in requirement of tuning of only a few parameters, and hence faster convergence. This process of using a pre-trained network \mathcal{N}_1 as the initialization for achieving a different but related task is referred to as transfer learning. This technique has been used for classification of abnormalities in brain from magnetic resonance images using ResNet34 as the pre-trained network [59], computer aided detection problems [60] with CifarNet, AlexNet and GoogLeNet networks and for detection of cracks and distress on

pavements with the VGG-16 network [61].

In this section, we identified the impact of width and depth of a neural network on the characteristic equations of nodes in different layers of the network. Specifically, while wider networks exhibit more number of simple characteristic equations and hence activations, deeper networks exhibit more complex equations describing complex structures in the input space. We also identified a generic pattern of the complexity of the characteristic equation, as one builds a deep neural network, and how this is exploited in the form of transfer learning in the literature. In the following, we focus our attention on the final layer of the classifier that takes the features of the pre-final layer and identifies separating boundaries in the feature space.

3.2.3 From Features to Feature Spaces

We have seen that intermediate nodes of the neural network results in selective activation of input space and creation of complex patterns for classification. Independent activation of each of these nodes in the intermediate layers, results in transformation of the input space to generate intermediate *feature spaces* on which subsequent layers operate. The different operations performed on the input space, and the resultant feature spaces for different activation functions are shown in Figure 3.7. For this illustration, $\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} 0.5 & -0.2 \\ -0.2 & 0.5 \end{bmatrix}$,

$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$. The input space – a flat plane – is first *rotated* and *scaled* according to weights in the weight matrix \mathbf{W} . This is followed by *translation* of the rotated and scaled space according to the bias \mathbf{b} . Finally, the translated space is *transformed* according to the nonlinear activation function $g(\cdot)$ to produce a curved surface from the flat input space. Figure 3.7 also compares the transformation achieved with different activation functions. In a neural network with multiple layers, the feature space of one layer acts as the input space for the successive layer. In such a network, the input space gets progressively transformed at

every successive layer to generate intermediate feature spaces of which the final one is used for classification.

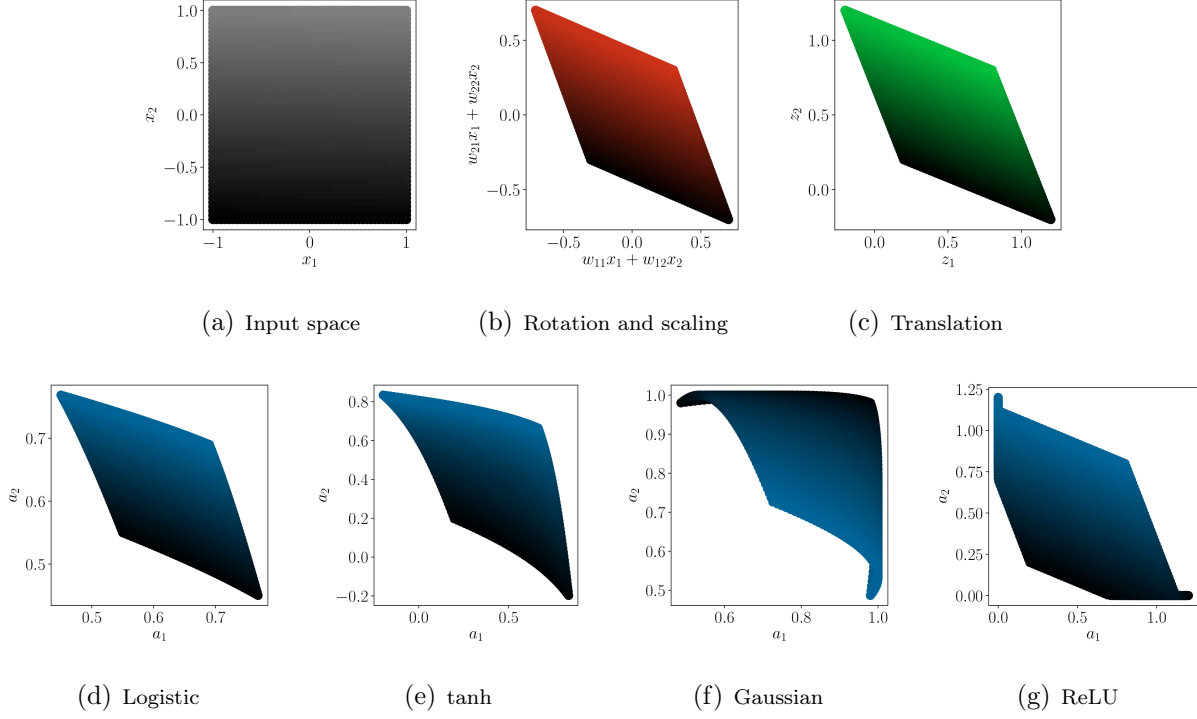


Figure 3.7: Example transformations of the input space to the feature space. The Input space (a) is first rotated and stretched (b), then translated to give the \mathbf{z} vector (c). Note how all the transformations are affine so far. The final nonlinear operation comes from the squishing action based on different activation function on the transformed space to give the final feature space representation (d),(e),(f) and (g). The dark regions in the input space are mapped to dark regions in the subsequent feature space.

3.2.4 The Final Layer: Separating Hyperplanes for Classification

The final layer of a neural network classifier solving a K -class classification problem typically employs the *softmax* activation that produces at the output, the probabilities of the input belonging to different classes. It must be noted that the *softmax* activation is a linear classifier, implying that it classifies different classes in *its* input space by identifying *hyperplanes* that (potentially) minimise the objective function. The number of nodes and activation function of the final layer of a neural network are therefore fixed, with only the

parameters of this layer playing a crucial role in identifying the separating boundaries. In order to study the properties of the weights of this layer, we consider a dataset with three classes that are linearly separable, and that are classified with a *softmax* classifier as shown in Figure 3.8.

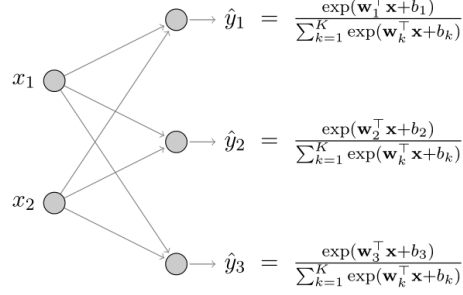


Figure 3.8: Sample architecture for linearly separable problems

The *softmax* classifier shown in Figure 3.8 (with $K = 3$) can be expressed as:

$$\hat{y}_j^{(i)} = \frac{\exp(\mathbf{w}_j^T \mathbf{x}^{(i)} + b_j)}{\sum_{k=1}^3 \exp(\mathbf{w}_k^T \mathbf{x}^{(i)} + b_k)} \quad (3.6)$$

The quantity $\mathbf{w}_j^T \mathbf{x} + b_j$ represents the distance of a point \mathbf{x} on the input space from the line $\mathbf{w}_j^T \mathbf{x} + b_j = 0$, i.e., the characteristic equation of the j^{th} output node. It can thus be observed that much like the feature extraction in the hidden layers, the classification performed in the final layer with a *softmax* activation also makes use of the distance of data points on the input space from the characteristic equation.

Figure 3.9a shows the data points belonging to the three classes highlighted in different colours, the lines representing the characteristic equations of each node in the *softmax* layer and the decision boundaries for the three classes identified by the network. The width of the lines representing the characteristic equations in Figure 3.9a are proportional to the norm of the corresponding weight vector consisting of two weights per class. The membership of each point on the input space to each class is characterized by its distance from the class's hyperplane. The arrows point in the direction of positive distance/membership. The greater

the membership of a point to a given hyperplane, the higher its probability of belonging to that class. For example, in Figure 3.9a the red line represents the equation $3x_1 - 3.3x_2 - 0.01 = 0$ and the arrow represents the region of the input space with positive distance from the red line. The region highlighted in red has the greatest membership with respect to the red line, in comparison to the other lines. Using Equation (3.6), we get the final probability vectors for all the points on the input space (Figure 3.9b). Notice how, due to the constraint that $\sum_{k=1}^3 \hat{y}_k = 1$, the final probability vector lies on the diagonal plane. In fact, for K -dimensional classification problem, when one uses a *softmax* activation in the final layer, the resultant vector will lie on a K -dimensional diagonal *hyperplane*.

From the discussion so far, we can say that if a point \mathbf{x} in the input space, belongs to cluster c , then,

$$\mathbf{w}_c^\top \mathbf{x} + b_c > \mathbf{w}_k^\top \mathbf{x} + b_k \quad k \neq c$$

It follows that the decision boundary between any two classes i and j can be identified using the hyperplanes of said classes, as the set of points \mathbf{x} where the memberships are equal, i.e.,

$$\begin{aligned} \mathbf{w}_i^\top \mathbf{x} + b_i &= \mathbf{w}_j^\top \mathbf{x} + b_j \\ (\mathbf{w}_i - \mathbf{w}_j)^\top \mathbf{x} + (b_i - b_j) &= 0 \end{aligned} \tag{3.7}$$

As a result, the final layer exhibits severe degeneracy of parameters, which are described in the following.

3.2.5 Degeneracy of parameters using softmax activation

Translation degeneracy: Consider a set of parameters (\mathbf{W}, \mathbf{b}) that can accurately classify the data. Consider vector \mathbf{w}_0 and scalar b_0 , $\mathbf{w}_0 \in \mathbb{R}^m, b_0 \in \mathbb{R}$. Then the translated set of parameters $(\mathbf{W} + \mathbf{W}_0, \mathbf{b} + \mathbf{b}_0)$, where $\mathbf{W} + \mathbf{W}_0 = \begin{bmatrix} \mathbf{w}_1 + \mathbf{w}_0 & \mathbf{w}_2 + \mathbf{w}_0 & \cdots & \mathbf{w}_K + \mathbf{w}_0 \end{bmatrix}^\top \in \mathbb{R}^{K \times m}$

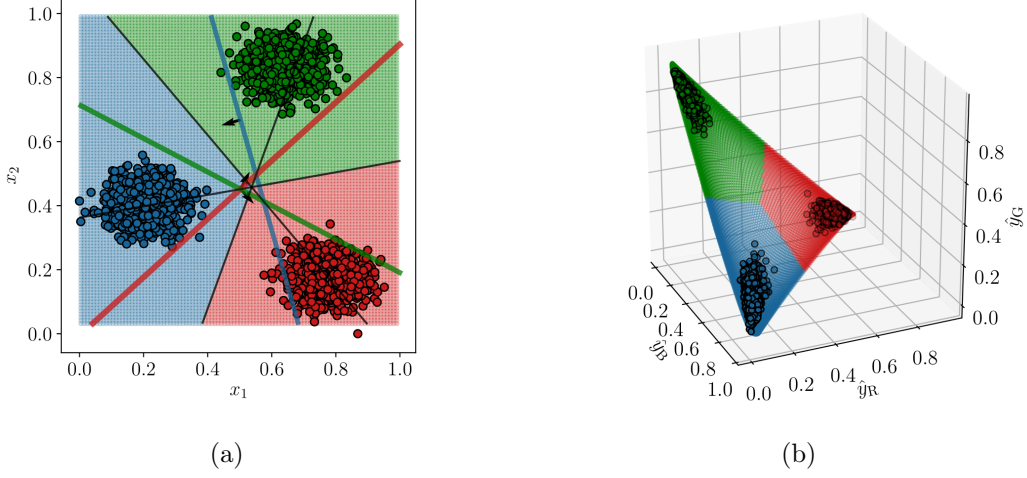


Figure 3.9: (a) Final estimated segregation on the input space. Thickness indicates the magnitude of \mathbf{w}_j for class j . The black lines are constructed using the boundary equation—Equation (3.7). (b) Identified probabilities of each point on the input space

and $\mathbf{b} + \mathbf{b}_0 = \left[b_1 + b_0, \quad b_2 + b_0, \quad \dots, \quad b_K + b_0 \right]^\top$, will also classify the data with the same accuracy.

Proof: Let a data point $\mathbf{x}^{(i)}$ belong to class c_i . Then, $\forall i \in \{1, \dots, N\}$ that are correctly classified by the classifier,

$$\begin{aligned}
 \mathbf{w}_{c_i}^\top \mathbf{x}^{(i)} + b_{c_i} &> \mathbf{w}_k^\top \mathbf{x}^{(i)} + b_k, \quad k \neq c_i \\
 \implies \mathbf{w}_{c_i}^\top \mathbf{x}^{(i)} + b_{c_i} + \mathbf{w}_0^\top \mathbf{x}^{(i)} + b_0 &> \mathbf{w}_k^\top \mathbf{x}^{(i)} + b_k + \mathbf{w}_0^\top \mathbf{x}^{(i)} + b_0, \quad k \neq c_i \\
 \implies (\mathbf{w}_{c_i} + \mathbf{w}_0)^\top \mathbf{x}^{(i)} + (b_{c_i} + b_0) &> (\mathbf{w}_k + \mathbf{w}_0)^\top \mathbf{x}^{(i)} + (b_k + b_0), \quad k \neq c_i
 \end{aligned}$$

The decision boundary estimated using Equation (3.7), remains the same. In the event of \mathbf{w}_0 being a zero vector, one is essentially translating each of the class hyperplanes by the same magnitude, and the overall effect of this is nullified. In fact it can be seen that the translated weights and biases give the same set of output probabilities as the untranslated

weights and biases, and hence, loss value.

$$\begin{aligned}
\hat{y}_j(\mathbf{W} + \mathbf{W}_0, \mathbf{b} + \mathbf{b}_0) &= \frac{\exp((\mathbf{w}_j + \mathbf{w}_0)^\top \mathbf{x} + (b_j + b_0))}{\sum_{k=1}^K \exp((\mathbf{w}_k + \mathbf{w}_0)^\top \mathbf{x} + (b_k + b_0))} \\
&= \frac{\exp(\mathbf{w}_j^\top \mathbf{x}^{(i)} + b_j) \exp(\mathbf{w}_0^\top \mathbf{x} + b_0)}{\sum_{k=1}^K \exp(\mathbf{w}_k^\top \mathbf{x}^{(i)} + b_k) \exp(\mathbf{w}_0^\top \mathbf{x} + b_0)} \\
&= \frac{\exp(\mathbf{w}_j^\top \mathbf{x} + b_j)}{\sum_{k=1}^K \exp(\mathbf{w}_k^\top \mathbf{x} + b_k)} \\
&= \hat{y}_j(\mathbf{W}, \mathbf{b})
\end{aligned}$$

Multiplicative degeneracy: If a set of parameters (\mathbf{W}, \mathbf{b}) can accurately classify the data, the the set of parameters $(\alpha \mathbf{W}, \alpha \mathbf{b})$, $\alpha > 0$ will also classify the data with the same accuracy.

Proof: $\forall i \in \{1, \dots, N\}$ that are correctly classified by the classifier,

$$\begin{aligned}
\mathbf{w}_{c_i}^\top \mathbf{x}^{(i)} + b_{c_i} &> \mathbf{w}_k^\top \mathbf{x}^{(i)} + b_k, \quad k \neq c_i \\
\implies \alpha \mathbf{w}_{c_i}^\top \mathbf{x}^{(i)} + \alpha b_{c_i} &> \alpha \mathbf{w}_k^\top \mathbf{x}^{(i)} + \alpha b_k, \quad k \neq c_i, \alpha > 0
\end{aligned}$$

By a similar argument for the examples that are incorrectly classified, it follows that the accuracy of the classifier remains the same.

Additive degeneracy: If a set of parameters (\mathbf{W}, \mathbf{b}) and $(\mathbf{W}', \mathbf{b}')$ can accurately classify the data, the the set of parameters $(\mathbf{W} + \mathbf{W}', \mathbf{b} + \mathbf{b}')$, will also classify the data with the same accuracy. $\forall i \in \{1, \dots, N\}$ that are correctly classified by both the classifiers,

$$\begin{aligned}
\mathbf{w}_{c_i}^\top \mathbf{x}^{(i)} + b_{c_i} &> \mathbf{w}_k^\top \mathbf{x}^{(i)} + b_k, \quad k \neq c_i \\
\mathbf{w}'_{c_i}^\top \mathbf{x}^{(i)} + b'_{c_i} &> \mathbf{w}'_k^\top \mathbf{x}^{(i)} + b'_k, \quad k \neq c_i \\
\implies (\mathbf{w}_{c_i} + \mathbf{w}'_{c_i})^\top \mathbf{x}^{(i)} + (b_{c_i} + b'_{c_i}) &> (\mathbf{w}_k + \mathbf{w}'_k)^\top \mathbf{x}^{(i)} + (b_k + b'_k), \quad k \neq c_i
\end{aligned}$$

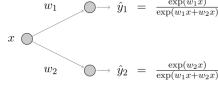
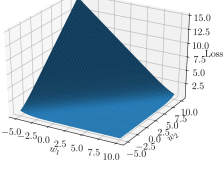
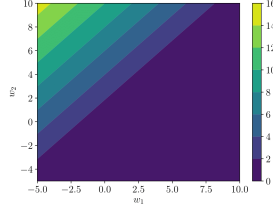
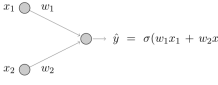
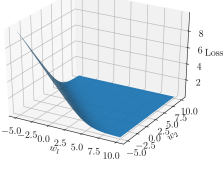
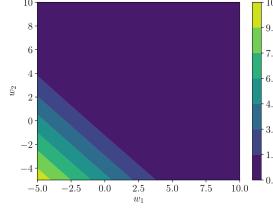
Superposition degeneracy: Combining the additive and multiplicative degeneracy, we

can see that if a set of parameters (\mathbf{W}, \mathbf{b}) and $(\mathbf{W}', \mathbf{b}')$ can accurately classify the data, the set of parameters $(\alpha\mathbf{W} + \beta\mathbf{W}', \alpha\mathbf{b} + \beta\mathbf{b}')$ where $\alpha, \beta > 0$, will also classify the data with the same accuracy.

Table 3.2 shows the impact of degeneracy of parameters as a function of the weights for two different architectures. It can be observed from Table 3.2 that the loss decreases sharply for only a limited range of weights and remains relatively flat for majority of the configurations of weights. It therefore follows that a network that is being trained with the dataset \mathcal{D} can witness decreasing losses solely due to the increase in value of the weights in the final layer. However, this does not involve any modification of the decision boundaries, and hence any improvement in the accuracy of classification. In this context, the technique of weight normalization and regularization [62, 63], that restricts the norm of the weight vector incoming on a single node to a value of one, and early stopping [64, 65] prevents such a scenario. It must be noted here that the impact of degeneracy of parameters on the input space of a layer is different for the hidden and final layers. Specifically, while the degenerate weights in the hidden layer result in different selective activations of the input space and hence different features extracted from the input, the degenerate weights in the final layer do not result in any change in the decision boundaries, and can provide a false sense of *better learning*.

In summary, we identified different design choices in building neural networks, including the number of nodes, number of layers and activation function that influence the performance of the network. We studied the contribution of each of these factors to the final output of the network with a systematic analysis of the role of each node, the number of layers and the number of nodes in the following. In the following, we discuss how these individual components are combined to achieve a particular classification task.

Table 3.2: The nature of the Loss function exhibiting degeneracy of parameters

Label	Architecture	Loss landscape	2 D projection Loss landscape
a			
b			

3.3 How does a Neural Network Learn the Mapping?: From Parts to Whole

In this section we merge the different components of the network and show how the network activates different segments from the input space, converting a nonlinear classification problem into a linear one. We consider three illustrative examples that highlight these features and an application of classification to fault diagnosis in a continuously stirred tank reactor. Figure 3.10 shows the dataset for three different examples, i.e., two interlocked *moons*, two concentric *circles* and two *spirals*. In each case, we varied the number of nodes in hidden layers (n), number of hidden layer ($H = L - 1$) and the activation function ($g(\cdot)$) and trained networks with different combinations. The architectural details, classification loss and accuracy for all networks trained on the three examples are listed in Table 3.3. In each case, the network was trained with 80% of the dataset and tested on 20% of the dataset. Tuning of parameters was carried out with a momentum based gradient descent algorithm, i.e., ADAM implemented in `Keras`. It can be observed from Table 3.3 that the performance of the networks are highly sensitive to the architecture as well as the complexity of the problem. In order to further understand how the behaviour of the network as a whole is constituted from the individual elements discussed in the previous section, we study the

transformation of feature space and selective activation performed by different networks.

We study the network with three hidden layers ($H = 3$) and three nodes in each hidden layer ($n = 3$) for the three illustrative examples. Figures 3.11, 3.12 and 3.13 show the feature space of the hidden and final layers of the networks trained on the moons dataset with *tanh*, *Gaussian* and *ReLU* as the activation function respectively. In each case, the interlocked moons can be observed to be untangled and progressively separated by successive layers of the network. It can be observed that for the same architecture, the three activation functions achieve the same objective by untangling the moons in different manners. For example, while the *tanh* and *Gaussian* activation functions can be observed to transform the input shape in a manner that clearly separates the two classes at the input to the third hidden layer, the network with *ReLU* as the activation function is unable to achieve adequate segregation between the classes. It can also be observed from Figures 3.11 and 3.12 that the transformation achieved by these networks at the input to the third hidden layer allows identifying a separating hyperplane in the $\mathbf{z}^{[3]}$ -space for the networks. However, the third layer of the networks further transforms the space to increase the separation between the classes, and hence result in a lower value of loss (at the same classification accuracy). It can be inferred from these observations that increasing the depth of a network beyond a certain threshold, although yielding a decrease in the loss, does not necessarily imply an improvement in the internal representation of the network. Figures 3.14, 3.15 and 3.16 show the feature space of the layers of a network trained on the circle dataset with *tanh*, *Gaussian* and *ReLU* as the activation function respectively. It can be observed that the same network architecture now results in a different transformation of the input space that results in the separation of the two concentric circles. In addition, the two classes can be observed to be separable at the input to the third layer (for the networks with *tanh* and *Gaussian* as the activation function) while the additional layer serves to increase the distance between the two classes, thereby resulting in a lower loss.

It can be observed from Table 3.3 that networks trained on the spiral dataset require

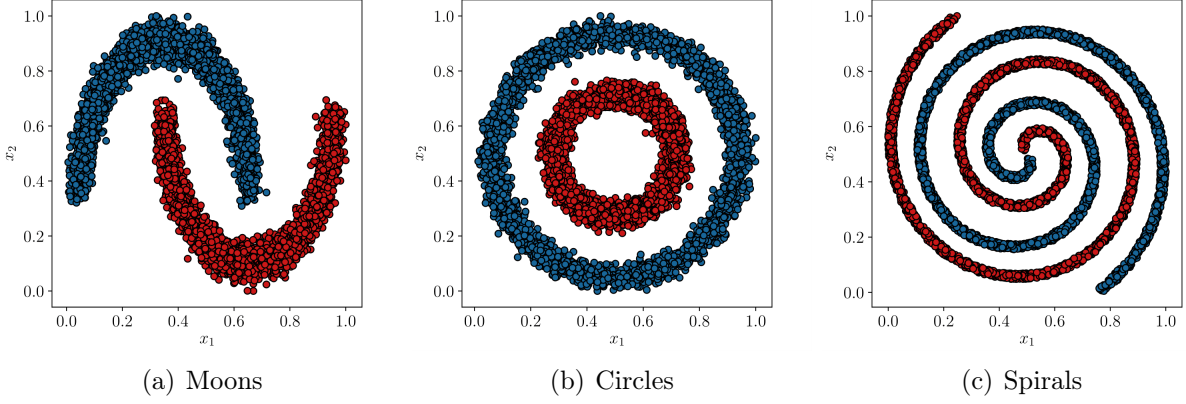


Figure 3.10: Example datasets for studying the operation of neural networks

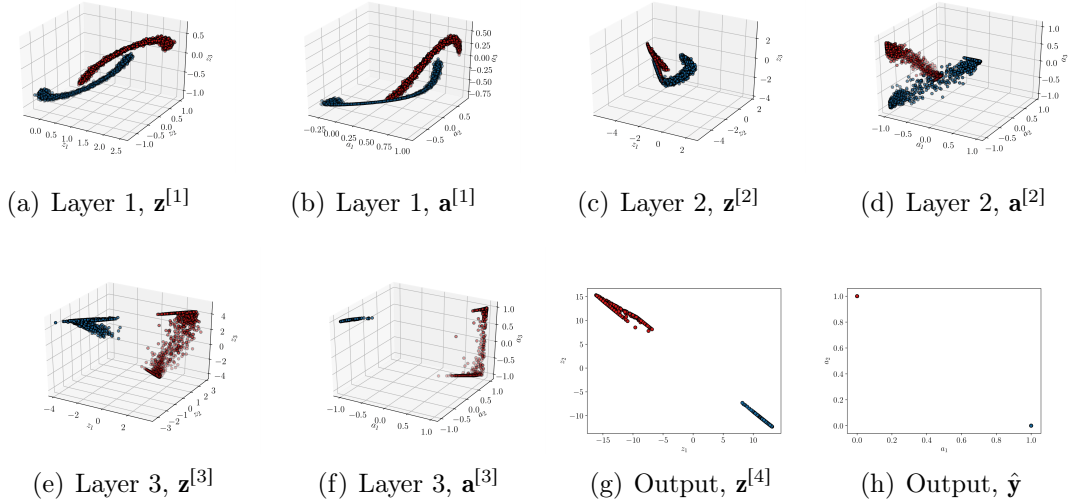


Figure 3.11: Internal internal representation of network with $n = 3, H = 3$ and \tanh as the activation function trained on the moons dataset

more resources than those trained on the moons and circles dataset for approximately same levels of performance. This is because of the highly intertwined nature of the dataset that results in a highly nonlinear separating boundary between the classes. Figures 3.17, 3.18 and 3.19 show the node-sensitive selective activation of the networks with $n = 10, H = 10$ and \tanh , *Gaussian* and *ReLU* as the activation function respectively. Each plot in Figures 3.17, 3.18 and 3.19 represents the selective activation performed by one node in the network, with the columns representing different layers and rows representing different nodes in one

Table 3.3: Consolidated results of different architectures after 1000 epochs of training

Example	Activation	H	n	Loss	Accuracy
Moons	tanh	1	3	0.14	93 %
		1	5	3×10^{-4}	100 %
		2	3	3.6×10^{-7}	100 %
		3	3	1.1×10^{-7}	100%
	ReLU	3	3	0.24	88 %
		3	5	0.2	90 %
	Gaussian	3	3	6×10^{-5}	100%
Circles	tanh	1	3	0.46	78 %
		1	5	2.2×10^{-3}	100 %
		2	3	4×10^{-7}	100%
		3	3	2.9×10^{-7}	100%
	ReLU	3	3	5×10^{-4}	100 %
		3	3	5×10^{-4}	100 %
Spirals	tanh	1	3	0.62	59%
		1	10	0.65	61%
		1	50	0.65	61%
		1	100	0.61	61%
		3	10	0.61	60%
		3	100	0.65	61 %
		10	10	10^{-5}	100%
		10	10	3×10^{-4}	100%
	ReLU	10	10	9×10^{-4}	100%
		3	100	6×10^{-5}	100%

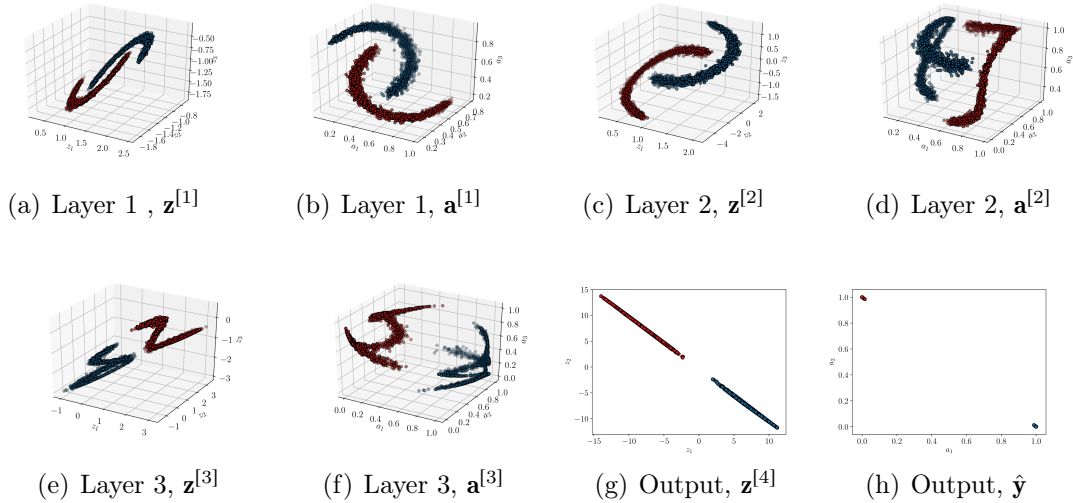


Figure 3.12: Internal internal representation of network with $n = 3, H = 3$ and *Gaussian* activation function trained on the moons dataset

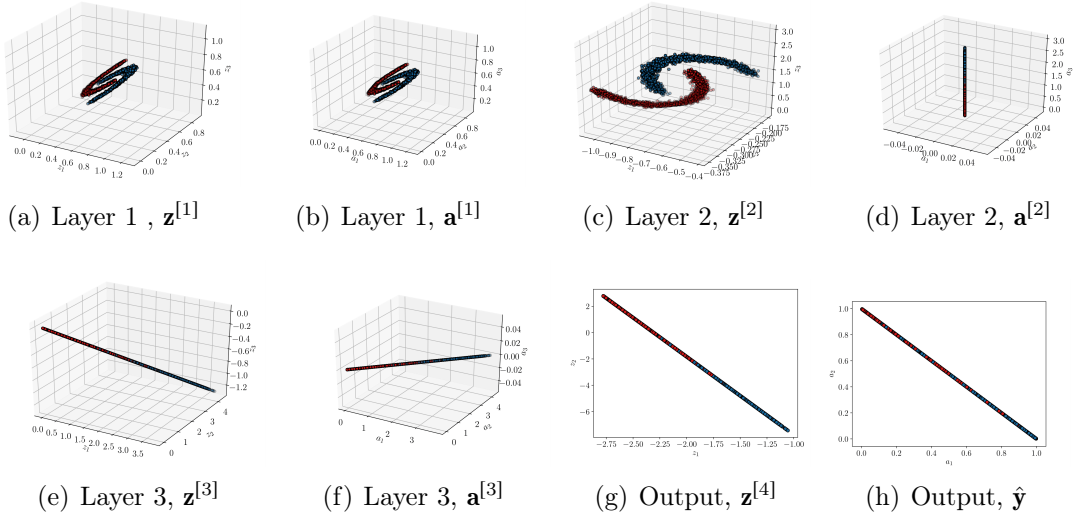


Figure 3.13: Internal representation of the network with $n = 3, H = 3$ and $ReLU$ as the activation function trained on the moons dataset

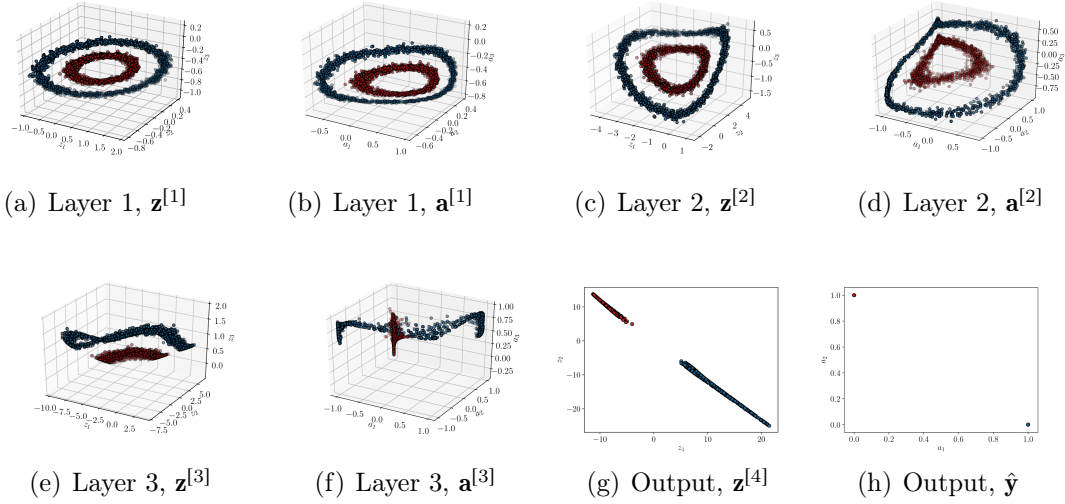


Figure 3.14: Internal representation of network with $n = 3, H = 3$ and \tanh as activation function trained on circles dataset

layer. It can be observed from Figures 3.17, 3.18 and 3.19 that the nodes in initial layers of the network identify simple patterns on the input space, while the final layers operate on these simple patterns to generate complex shapes, most of which closely resemble the spiral distribution of the data in the input space.

Figure 3.20 compares the selective activation performed by the final hidden layer of the

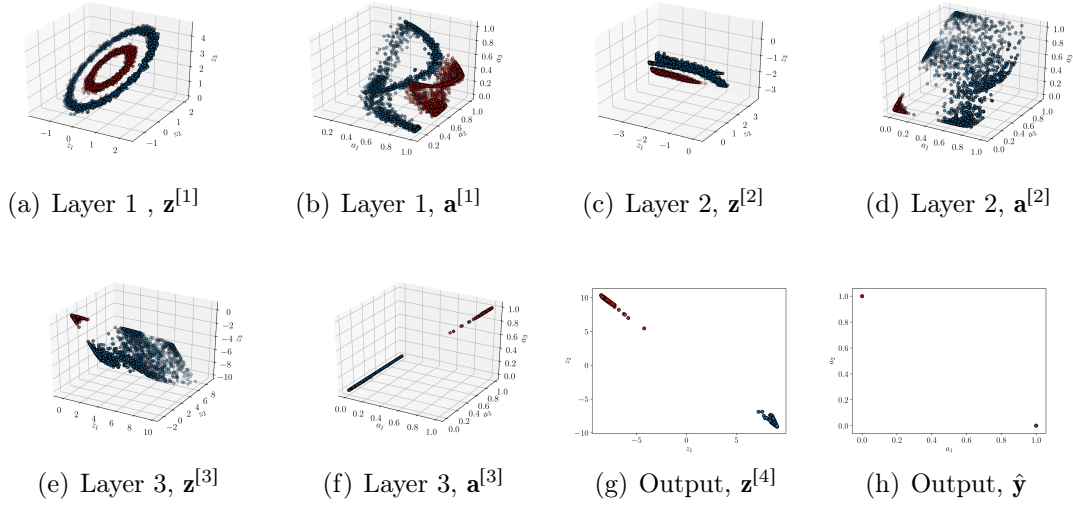


Figure 3.15: Internal representation of network with $n = 3, H = 3$ and *Gaussian* activation function trained on the circles dataset

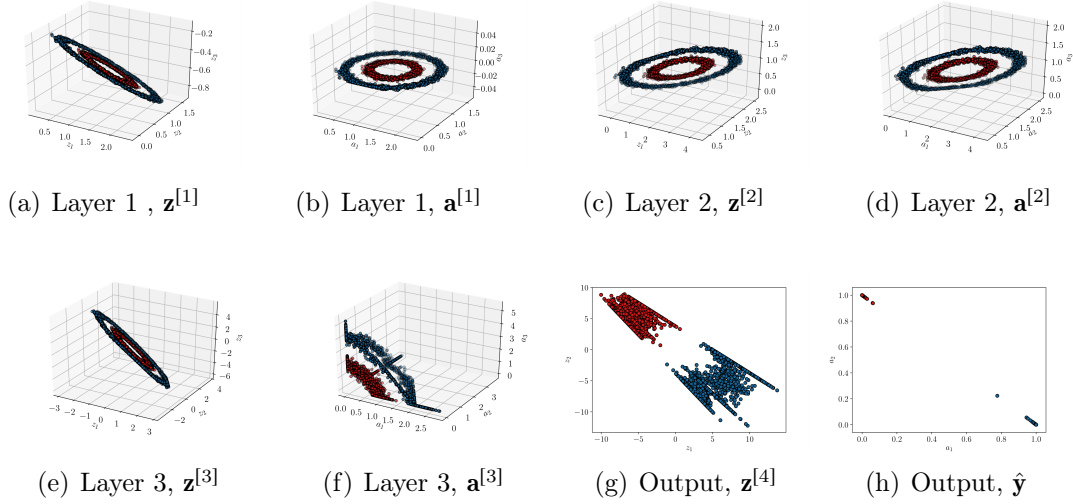


Figure 3.16: Internal representation of network with $n = 3, H = 3$ and *ReLU* as the activation function trained on the circles dataset

networks with $n = 10, H = 10$ and $n = 100, H = 3$, which highlights the difference between the two representations. Specifically, final hidden layer of the deep network reveals complex geometrical patterns constructed on the input space, while the shallow and wide network identifies very simple patterns on the input space. This verifies the observation made in the earlier sections regarding the impact of depth and width of the network on the complexity

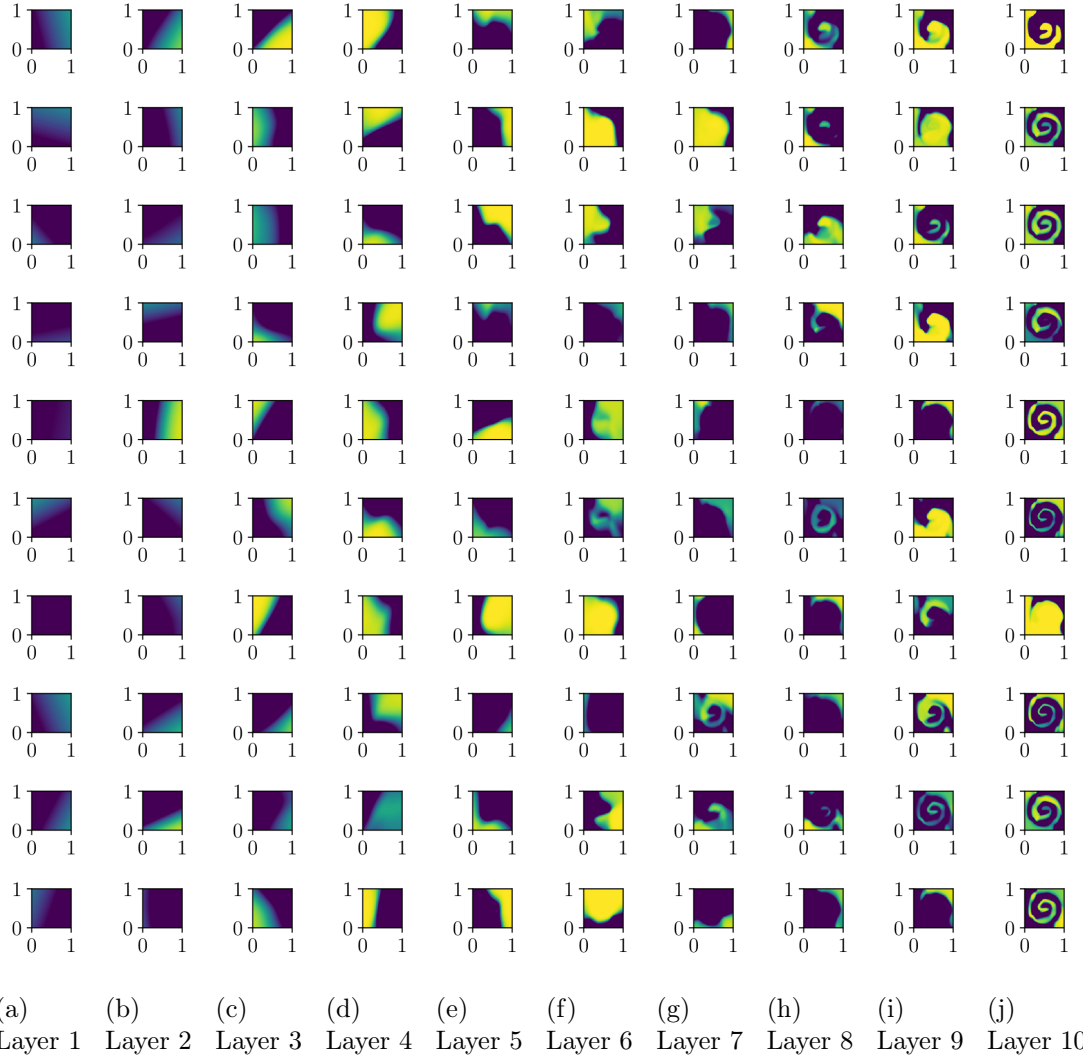


Figure 3.17: Node-specific activation of input space for the network with $n = 10, H = 10$ and \tanh as the activation function trained on the spirals dataset. Blue denotes activation of -1 , and yellow denotes activation of 1

of features extracted by the network.

The above discussion highlighted the transformation of input space performed by each layer in the network as well as the selective activation performed by the individual nodes in a layer. It was observed that all the nodes in the network learn (tune their parameters) towards a common objective of minimising the loss and in the process (i) transform the space and (ii) identify selective activations to separate the two classes. We also highlighted the influence of width and depth on the internal representation of the network. We next

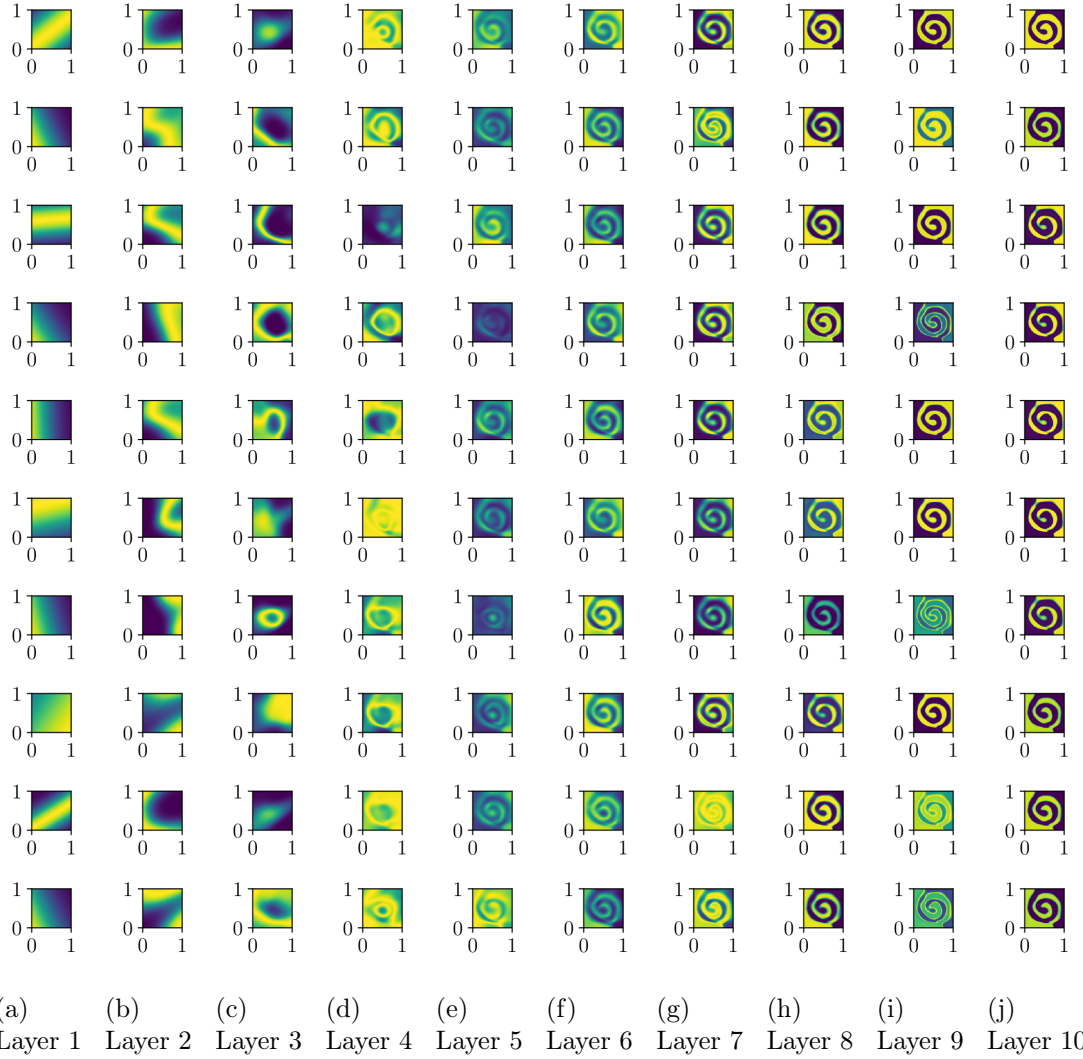
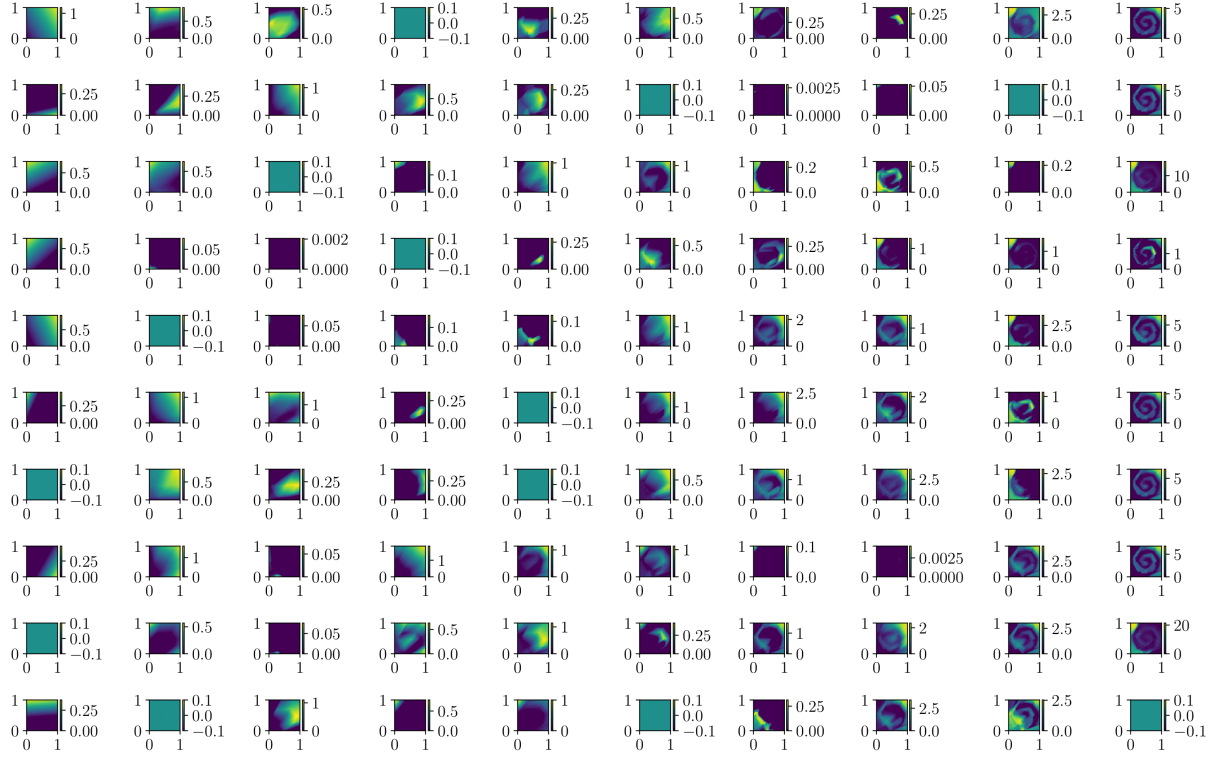


Figure 3.18: Node-specific activation of input space for the network with $n = 10, H = 10$ and *Gaussian* activation function trained on the spirals dataset. Blue denotes activation of 0, and yellow denotes activation of 1

discuss the application of classification to fault diagnosis in Chemical plants and examine the feature spaces and activations identified by the neural networks.

One of the most common applications of classification in chemical engineering involves diagnosing the operation of the plant as being normal or abnormal. and has been extensively studied in the literature [66, 67, 68, 69, 70, 71, 72, 73, 74, 75]. This involves employing the measurements obtained from sensors installed in the plant and identify the same to belong one of many classes with each class representing a state of operation. The measurements



(a) Layer 1 (b) Layer 2 (c) Layer 3 (d) Layer 4 (e) Layer 5 (f) Layer 6 (g) Layer 7 (h) Layer 8 (i) Layer 9 (j) Layer 10

Figure 3.19: Node-specific activation of input space for the network with $n = 10, H = 10$ and *ReLU* as the activation function trained on the spirals dataset.

from sensors along with appropriate labels therefore constitute the dataset used for training neural network based classifier.

It is useful to understand how the neural network internally represents and classifies the different normal and abnormal regions - i.e., peer into the black-box to gain some insights. The earliest work in this regard in the chemical engineering literature was by Vaidhyanathan and Venkatasubramanian [76] who studied the fault space classification structure of a CSTR process. We continue that effort further here to gain more insights by using the more modern tools available now.

We consider a exothermic first order continuously stirred tank reactor, with temperature

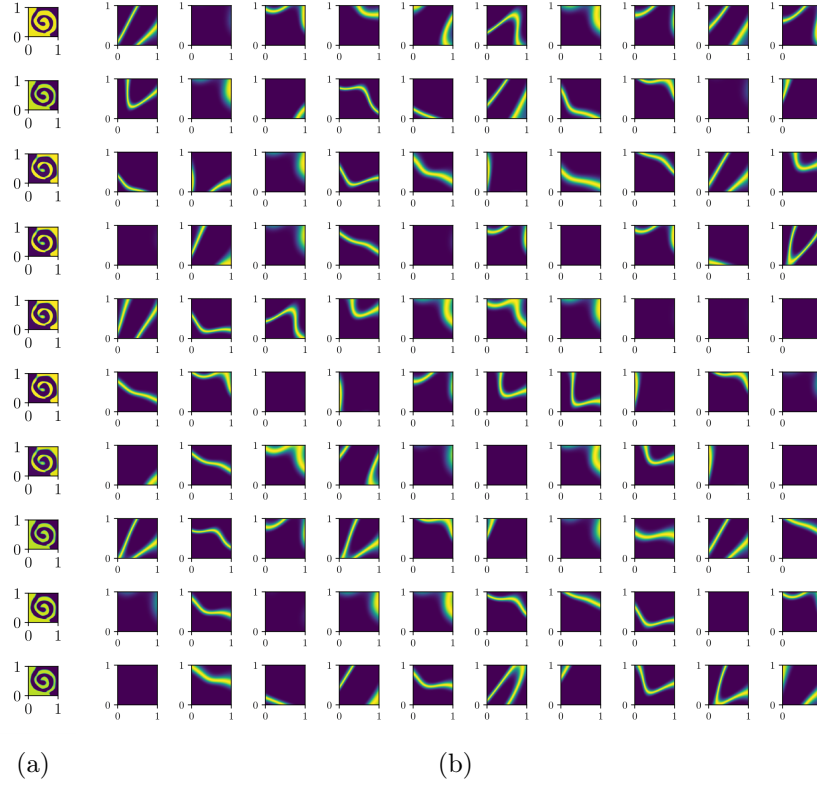


Figure 3.20: A comparison of the final hidden layer activation for different architectures with Gaussian activation in example 3: Architecture (a) $n = 10$, $H = 10$ (b) $n = 100$, $H = 3$. Blue regions denote area in the input space where activation of the node becomes 0. Yellow denotes activation of 1. Notice how deeper networks yield complex activations while, shallow-wide networks have relatively simple activations.

control, that operates according the following equations:

$$\begin{aligned}
 \text{Concentration} \quad \frac{dC}{dt} &= \frac{Q}{V} (C_{\text{in}} - C) - kC \exp\left(-\frac{E_a}{RT}\right) \\
 \text{Temperature} \quad \frac{dT}{dt} &= \frac{Q}{V} (T_{\text{in}} - T) + \frac{-\Delta H_{\text{rxn}}}{\rho c_p} kC \exp\left(-\frac{E_a}{RT}\right) - \frac{UA}{\rho c_p V} (T - T_c) \\
 \text{Coolant temperature} \quad \frac{dT_c}{dt} &= \frac{Q_c}{V_c} (T_{c,\text{in}} - T_c) + \frac{UA}{\rho_c c_{p,c} V_c} (T - T_c)
 \end{aligned}$$

and that has two measurements – concentration of feed C_{in} and temperature of the inlet T_{in} that can be used to perform fault diagnosis. The process is simulated using the model in [77] with disturbances in the feed concentration as well as inlet temperature under normal operating conditions. The following faults are then introduced into the reactor (in addition

to disturbances) by changing the appropriate parameters of the system:

1. Normal operation
2. Fault 1: High inlet concentration
3. Fault 2: Low inlet concentration
4. Fault 3: High inlet temperature
5. Fault 4: Low inlet temperature
6. Fault 5: Low inlet concentration with moderately low inlet temperature

We generated 1200 samples of data for each class and performed training with 70% of the data, which was first used to train an input-output network that has no hidden layers. Figure 3.21 shows the normalised data of concentration and temperature belonging to different classes and the separating boundaries of the input-output-network. It can be observed from Figure 3.21 that, as expected, the input-output network identifies linear boundaries that separate the different classes, and deliver an accuracy of 96%. Since we now know that deep networks tend to make complicated features, and shallow-wide networks tend to exhibit a more varied set of final representation, we want to observe a combined effect of changing the width and depth simultaneously. We hence, trained networks with $n = 3, H = 3$ and $n = 6, H = 1$ and *tanh*, *Gaussian* and *ReLU* as activation functions to study the combined impact of depth and width, for each activation function on the performance of the network. The performance of all the models are listed in Table 3.4 while the separating boundaries identified by the different networks are highlighted in Figure 3.22 for the deep and shallow networks with different activation functions. It can be observed from Figure 3.22 that the networks with *tanh* as the activation function tend to identify simple and reasonable separating boundaries, while the networks with *Gaussian* activation function identify fairly complex boundaries. The boundaries identified by the network with $n = 3, H = 3$ and *ReLU* as the activation function also exhibit a similar behavior. In addition, it can also be

observed that the networks with *Gaussian* activation function result in disjoint segments of the input space being identified as belonging to the same class. It is noteworthy here that all these boundaries result in a classification accuracy of at least 98.7% and yet exhibit internal representations that are neither an accurate representation of the truth, nor robust to noise in the data. For example, one can easily identify regions in the input space which do not belong to the normal class (shaded in red) which will however be classified with exceedingly high probabilities as representative of healthy operation of the reactor. These regions and the corresponding data points constitute *adversarial examples* [78, 79, 80, 81] that pose a significant threat to the reliability of the network. This is in contrast with the input-output network and the networks with *tanh* as the activation function that do not exhibit such behaviours.

Table 3.4: Fault diagnosis example – Consolidated results after 1000 epochs of training on the test sample

Activation	H	n	Loss	Accuracy
-	0	0	0.1	96%
tanh	3	3	0.01	99%
Gaussian	3	3	0.03	98.8%
ReLU	3	3	0.009	99.7%
tanh	1	6	0.03	98.8%
Gaussian	1	6	0.03	99%
ReLU	1	6	0.037	98.7%

3.4 Major Results

One of the primary results in this chapter is the degeneracy in parameters using softmax activation function for classification. While degeneracy has been shown to exist in neural networks by virtue of nonlinear operations and not due to the choice of the loss-function [82], we show that parameter degeneracy occurs even in the final layer of softmax function. We have identified that in the final layer for classification, one can have – translation degeneracy, multiplicative degeneracy, additive degeneracy, and superposition degeneracy in parameters.

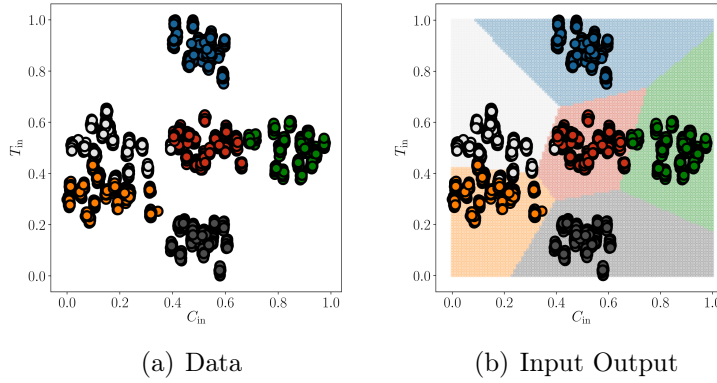


Figure 3.21: (a) Fault space data (b) Fault space classification with a simple input output model

There has been general consensus in literature that deeper networks outperform shallow neural networks, particularly for classification problems. This is shown in the case of residual neural networks [83]. Further investigation showed that shallow but wide networks can perform better in the classification task [84]. However the principle guiding the two is unclear. To look at the underlying feature representations our model test cases were simple for visualization purposes, without a residual skip connection.

We see that, there is an inherent trade-off between depth and width of neural networks. Similar performance can be obtained by increasing the depth or the width of neural networks. The difference however lies in the internal representation of the features. While increasing depth leads to more complex hidden representations deeper into the network due to nonlinear operations, increasing width combines many simpler features into complex ones.

We also note that for classification tasks, while generally increasing depth leads to better performance, one does require enough width of the neural network to exploit the depth. Changing the intermediary layer activation functions could also result in faulty feature-space classification, as shown using the CSTR example, for the same network architecture.

In conclusion, this chapter is an initial step towards a systematic approach towards the understanding of the internal mechanism of deep neural networks, to uncover key hidden properties that can be exploited to improve such networks. Future work in this direction

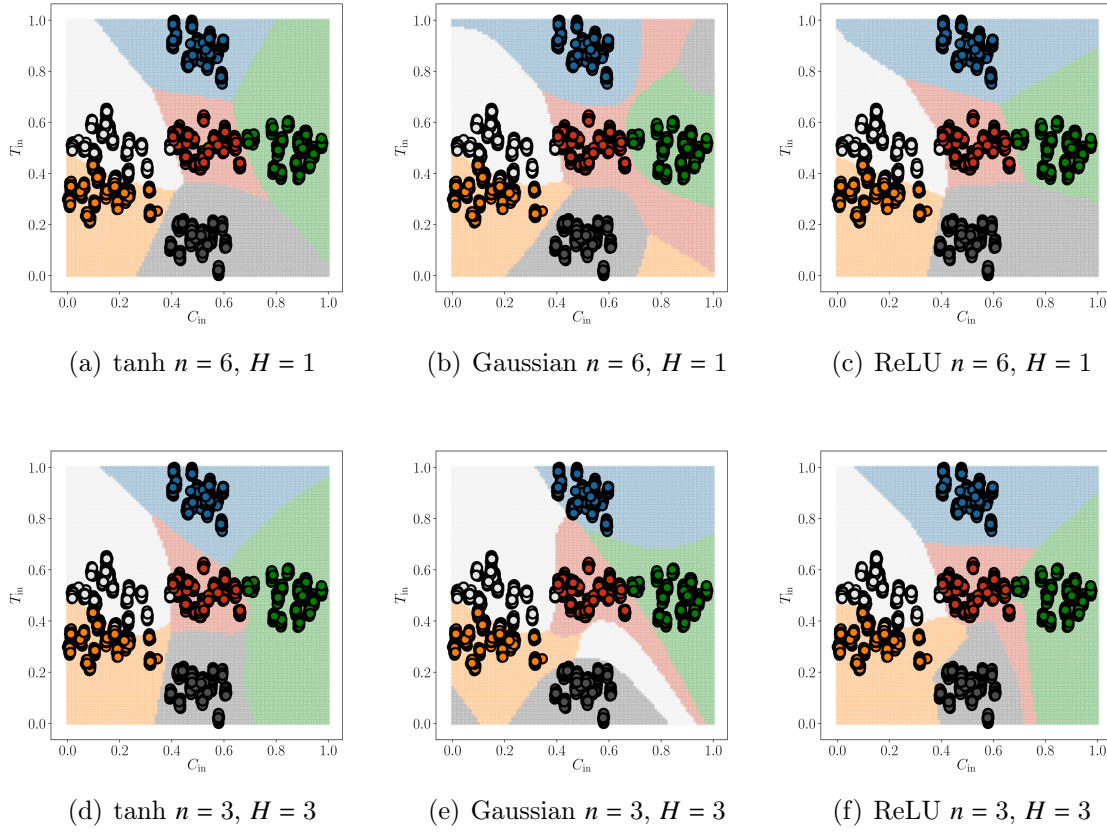


Figure 3.22: Fault space classification for different activation functions and architectures

would involve understanding how neural networks perform regression with internal representations. The goal is to elicit key insights for modeling engineering systems with neural networks in a manner that allows one to assign physical meaning to the internal representations and/or to be able to combine them with first-principles knowledge.

Notation

Vectors are bolded lower case. Matrices are bolded upper case. Elements not bolded are scalars.

If superscript $(i)/[l]$ are absent the quantity is assumed to be for the corresponding sample/layer respectively.

N number of samples in the dataset

m	dimension of the input
K	dimension of the output
L	number of layers in the network (excluding input and output)
H	number of hidden layers in the network ($= L - 1$)
n_l	number of nodes in layer l of the neural network
$\mathbf{x}^{(i)}$	$\in \mathbb{R}^m$ i^{th} input sample, $\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}, \dots, x_M^{(i)}]$
$\mathbf{y}^{(i)}$	$\in \mathbb{R}^K$ i^{th} output sample, $\mathbf{y}^{(i)} = [y_1^{(i)}, y_2^{(i)}, \dots, y_K^{(i)}]$
$\hat{\mathbf{y}}^{(i)}$	$\in \mathbb{R}^K$, i^{th} predicted output
$\mathbf{w}^{[l]}$	$\in \mathbb{R}^{n_{l-1}}$, weights connecting a node in layer $l-1$ to layer l , $\mathbf{w}^{[l]} = [w_1^{[l]}, w_2^{[l]}, \dots, w_{n_{l-1}}^{[l]}]^\top$
$\mathbf{W}^{[l]}$	$\in \mathbb{R}^{n_l \times n_{l-1}}$ Weight matrix connecting layer $l-1$ to layer l , note that $\mathbf{W}^{[l]} =$ $\begin{bmatrix} -\mathbf{w}_1^{[l]\top} - \\ -\mathbf{w}_2^{[l]\top} - \\ \vdots \\ -\mathbf{w}_{n_l}^{[l]\top} - \end{bmatrix}$
$\mathbf{b}^{[l]}$	$\in \mathbb{R}^{n_l}$ bias vector of the nodes in layer l , $\mathbf{b}^{[l]} = [b_1^{[l]}, b_2^{[l]}, \dots, b_{n_l}^{[l]}]^\top$
$\mathbf{a}^{[l]}$	$\in \mathbb{R}^{n_l}$ activation vector of nodes in layer l , codomain dependent on the activation
$\mathbf{z}^{[l]}$	$\in \mathbb{R}^{n_l}$ weighted sum of the inputs to nodes in layer l
$g^{[l]}(\cdot)$	Activation function of layer l

Chapter 4: Neural Networks for Regression

Models in process engineering can broadly be classified as (i) First-principles models, (ii) Data-driven models, and (iii) Hybrid models. First-principles models primarily make use of principles of conservation such as mass, momentum and energy balances, and/or constitutive equations, the parameters of which are identified using optimization techniques. In the current age of high throughput data, one of the more prominent modeling frameworks is data-driven modeling, of which neural networks have been favored greatly for regression (also called function approximation) and classification tasks. In Part 1 of this series [85], we presented a systematic approach to study the internal representations of neural networks and showed how the hidden representations in deep neural networks transform with increasing depth and width of the network for a classification task. This chapter is an extension of that paper for the class of regression tasks that aims to shed light on the internal representations of neural networks for function approximation.

With the advent of AI, much effort has been devoted towards exploiting neural networks for regression problems such as system identification [86, 87], predicting structure-property relationships [88], predicting potentials [89, 90, 91], predicting chemical structures [92], experimental conditions and properties [93, 94, 95]. However, as explained in Part 1, developing a deep neural network model is more of an art than science. One faces a bewildering array of choices with respect to the architecture, activation functions, loss functions, initialization conditions, data segmentation, training regimens, etc. All these, if done incorrectly, can potentially lead to rather spurious results. One has to understand the underlying structure of the representations in order to make more informed choices, and exploit the features of neural network representations for identification and monitoring. This is the challenge we try to address in this two-part paper.

There have been attempts in the past to explain the functioning of a neural network, using visualization techniques [50, 96], rule extraction [51], and randomization approaches to give insights into the importance of nodes in a neural network [52]. More recently, information bottleneck analysis seems to give more insight into the functioning of a neural network [53, 54, 55]. While these methods have shown promise, they need to be extended to include capabilities such as providing explanations.

Further, some regression tasks may not be possible with simple architectures. For example, training of the neural network while traversing the complex loss function landscape (please see details in Part 1 of this series), can be difficult with simple gradient descent schemes [97]. We highlight several of these aspects of training a neural network in this chapter, by visualizing the hidden representations of simple regression tasks as case studies. These simple problems can be viewed as the equivalent of particle-in-a-box and simple harmonic oscillator in quantum mechanics. Such models, despite their simplicity, play a key role in understanding and demonstrating the underlying phenomena of the systems.

Cybenko [98] proved that any continuous function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ can be approximated to a reasonable degree using a sigmoidal functions such that $|f(x) - \hat{f}(x)| < \epsilon$ for any arbitrary $\epsilon > 0$, where $\hat{f}(x) = \sum_{i=1}^m v_i \sigma(w_i^\top x + b_i)$, $w_i \in \mathbb{R}^n, b_i \in \mathbb{R}$. Further, Hornik et al.[99], proved that multilayer feed forward networks are universal approximators, for any choice of activation. Multiple theoretical studies have since shown these approximation results for various class of problems [100, 101, 102, 103].

From a design perspective, however, it is important to get a good idea of an appropriate architecture. While earlier work prove existence of these solutions, training of such networks tend to be difficult. Further, effect of noise on the estimates of parameters are not accounted for. There has been considerable effort on algorithms for neural network training [104, 105, 106]. Though these algorithms need to be well-studied and extended, our focus in this paper is in understanding the hidden features of the network during the learning of a regression task.

The chapter is organized as follows: we provide a brief background of a neural network (readers are directed to Part 1 of this 2-part paper for more details), followed by a peek under the hood of a neural network (this involves a look at the loss function landscape for simple examples). The following section guides us towards estimating input-output relationship for a sinusoidal signal. An important domain for neural network models is to estimate potential energy of configurations of a system, as molecular dynamics simulation are computationally intensive. Hence, we share a demonstrative example of estimating an energy function landscape, followed by a detailed discussion and conclusion of the chapter.

4.1 Mathematical Background

4.1.1 Problem Formulation

In a regression problem, the dataset $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N$ consists of (N) examples ($i = 1, 2, 3, \dots, N$) of the regressors $\mathbf{x}^{(i)} \in \mathbb{R}^{n_x}$ and outputs $\mathbf{y}^{(i)} \in \mathbb{R}^{n_y}$ that are used to build a model of the underlying process. As opposed to classification problems discussed in the last chapter, the outputs $\mathbf{y}^{(i)}$ in regression problems represent continuous-valued measurements obtained from the system. The problem of parameter estimation in regression tasks is then formulated as an optimization problem, aimed at minimising the following objective function:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \|\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\|_2^2 \quad (4.1)$$

The squared error is typically adopted because of mathematical tractability and convenience, and a regression technique attempts to minimize the average of the squared error for all training examples.

A neural network \mathcal{N} designed to solve the above problem is described by a predefined architecture \mathcal{A} that is parameterised in Θ and that expresses the outputs as a function of the regressors as:

$$\hat{\mathbf{y}}^{(i)} = \mathcal{N}(\mathbf{x}^{(i)}, \Theta) \quad (4.2)$$

The architecture \mathcal{A} of a neural network \mathcal{N} encapsulates all the design choices that can be made by the user (organization of layers, activation function $g(\cdot)$, number of layers L and number of nodes in each layer $n_l, l = 1, 2, 3, \dots, L$) while the parameters Θ include the weights $\mathbf{W}^{[l]}$ and biases $\mathbf{b}^{[l]}$ of all layers $l = 1, 2, 3, \dots, L$. Once the architecture of a network is decided and fixed, the parameters are learnt via training through the iterative training algorithm. Further discussion on the architecture and parameters can be found in the last chapter. In the following, we briefly describe the operation of a neural network and present a simple regression example motivating the rest of the article.

4.1.2 Regression with Neural Networks

The task of regression, as discussed earlier, involves building a *model* of the underlying process/system that relates the inputs (regressors) to the outputs. Let us consider a generic multi-layer neural network \mathcal{N} that we intend to use for regression. Each layer of \mathcal{N} performs a (non)linear operation on its inputs, which are equal to the weighted combination of the outputs of the preceding layer as follows:

$$\mathbf{a}^{[l]} = g^{[l]}(\mathbf{z}^{[l]}) = g^{[l]}(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}) \quad \forall l \in \{1, 2, 3, \dots, L-1\} \quad (4.3)$$

where $\mathbf{a}^{[l]}$, $g^{[l]}(\cdot)$, $\mathbf{z}^{[l]}$, $\mathbf{W}^{[l]}$ and $\mathbf{b}^{[l]}$ represent the activation (output), activation function, input, weights and biases of the l^{th} layer of \mathcal{N} respectively, with $\mathbf{a}^{[0]} = \mathbf{x}$ and $\mathbf{a}^{[L]} = \hat{\mathbf{y}}$. In this work, we consider four different activation function, i.e., logistic function (σ), hyperbolic tangent (\tanh) and Rectified Linear Unit (ReLU) to study the influence of $g(\cdot)$ on the performance of the network. The output activation function is always considered to be a linear activation. As discussed earlier, several of above factors constitute design choices that are required to be identified by the user before training a network. It must be noted here that the performance of a given network depends on the distribution of the underlying data, and the complexity of the function relating the regressors and outputs. For example, data

that is generated according to the logistic function, such as the Verhulst-Pearl equation of population growth, can be expressed directly in the form of the logistic activation function, requiring a network with only one node with logistic activation function. On the other hand, a neural network with one of the above activation functions is likely to require far more resources to approximate the current-voltage data generated by an oscillator circuit. In general, however, because of intractable complexity of the distribution of data and/or the lack of a clear (qualitative and quantitative) understanding of the contribution of each of the factors to the performance of the network, these choices are often arrived at with (sometimes exhaustive) trial-and-error approach. In the following, we illustrate this phenomenon with an illustrative example, and motivate the rest of the chapter.

Let us consider a scenario in which the regressors and outputs are related as:

$$y = \sin(x) \quad x \in [-10, 10] \quad (4.4)$$

In order to train a neural network to capture the above relation, we generated 2000 samples of the data (without noise) and used a train-test split of 70% – 30%. We designed networks with 2, 5 and 10 nodes in each layer for a 5-layer networks with σ , *tanh* and *ReLU* as the activation functions. The activation of the final layer was set to be a linear function in all the configurations. We then trained the networks with the RMSPROP optimizer for 500 epochs with a batch size of 128 samples. All the studies in this article have been implemented in Python, with Keras. The testing data and predictions of trained neural networks are presented for each configuration in Figure 4.1. From our earlier discussion, one would expect the networks with smooth activation functions (σ and *tanh*) to perform better (because of the smooth nature of the underlying sinusoidal data) than the discontinuous *ReLU*. However, it can be observed that while the network with *tanh* as the activation perform well, those with σ are unable to approximate the sinusoidal function with the same amount of resources. Furthermore, the networks with *ReLU* as the activation function exhibit very high accuracy

of predictions. These non-obvious observations involved with training and evaluating neural networks, among other factors such as complexity of distribution of data often lead to a trial-and-error approach being adopted in several applications. The authors believe that it is the lack of a clear understanding of the internal mechanism of neural networks that leads to such observations and the trial-and-error approach. This article presents a systematic study of the internal mechanism, identifying the impact of each architectural design element towards the performance of the network as a whole for the class of regression problems. Much like the first part of the article, we begin with a small neural network that lends itself to a detailed analysis, and build wider and deeper networks to understand the impact of width and depth on the performance of a network.

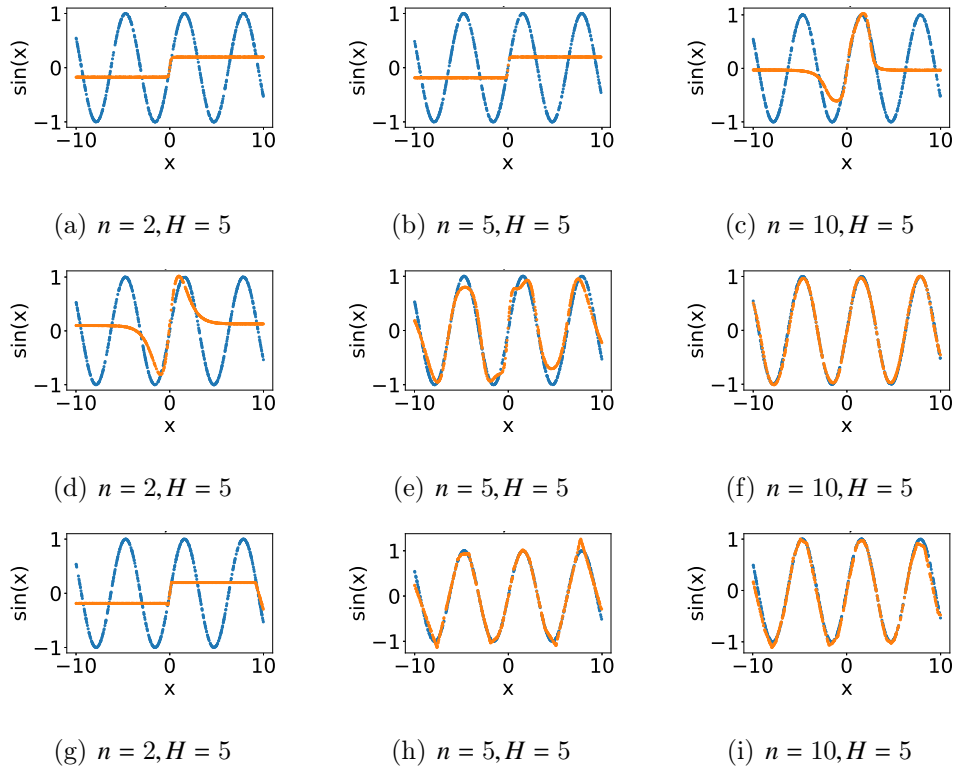


Figure 4.1: Testing data (blue) and model predictions (orange) for a 5-layer network with 2, 5 and 10 neurons in each hidden layer and σ (Row 1), \tanh (Row 2) and $ReLU$ (Row 3) as activation function

4.2 Peeking Under the Hood of a Neural Network

A neural network, in general, consists of L layers of n_l neurons in the l^{th} layer connected together in a manner determined by the architecture, such as fully connected and convolutional connections. In this article, we restrict ourselves to the networks that are fully connected, i.e., each neuron in a given layer is connected to all neurons in the immediately preceding layer. Each layer of the network performs a set of non-linear operations on the data and produces the input to the next layer, which are further processed in a recursive manner as described in Equation (4.3). We begin by studying the operations performed by individual nodes followed by layers in a network.

4.2.1 Node-specific Local Approximation

Let us consider a system that takes as input, a one-dimensional variable x and produces as output, the absolute value of the input as:

$$y = \text{abs}(x) \tag{4.5}$$

This function can be expressed in terms of the *ReLU* activation function (very commonly used in neural networks) as:

$$y = |x| = \text{ReLU}(x) + \text{ReLU}(-x) \tag{4.6}$$

so that a neural network with one input node, one hidden layer with two nodes and an output layer with one node as shown in Figure 4.2 can *exactly* learn the absolute function. In such a neural network, each neuron effectively learns the underlying function for a subset of inputs in such a manner that the sum of their outputs replicates the behaviour of y as a function of x . In other words, each neuron in the tiny network learns the underlying function in a local neighbourhood. To further illustrate this behaviour, let us consider one

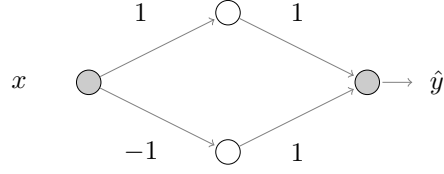


Figure 4.2: Representation of the absolute function generator with a neural network

cycle of a triangular wave function as shown in Figure 4.3, which can be expressed as:

$$y = \text{ReLU}(x) - 2\text{ReLU}(x - 0.25) + 2\text{ReLU}(x - 0.75) - \text{ReLU}(x - 1) \quad (4.7)$$

It can be now be seen that, owing to the limited number of nodes in the tiny network

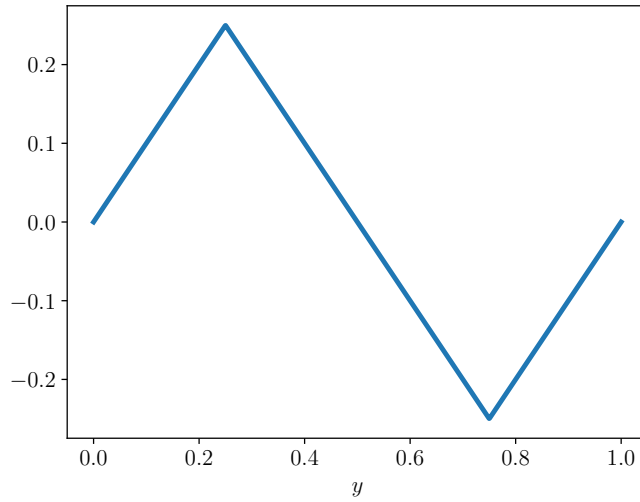


Figure 4.3: Triangular wave function

considered in Figure 4.2, it cannot be trained to learn this function completely. It can be observed from the above examples that while the absolute function could be expressed as the sum of two *ReLU* terms and thus learnt by a network with two nodes in the hidden layer, the triangular wave required four terms and would thus need at least four nodes in

the hidden layer. Therefore, functions that have complex structures – in relation to the activation function of a network – will require more neurons (and possibly layers) to be approximated adequately compared to functions that are relatively simple. Furthermore, the above examples illustrate the behaviour and contribution of each neuron in the network towards learning the underlying function. Specifically, each neuron in the network, based on the parameters associated to itself, identifies a local region in the domain of the function and learns the functional transformation within the region. In the process of training, the parameters of the neurons are adjusted with respect to each other in such a manner that the sum of the individual contributions results in underlying functional transformation being learnt.

Such an approach of expressing a signal as a sum of the contributions of several units, each with a functional transformation is akin to decomposition methods such as Fourier transform and Wavelet transform. However, it must be noted that while the transforms make use of basis functions that exhibit orthonormality, their equivalent functions, i.e., the activations of individual neurons do not, in general, exhibit any such properties. These constraints must be explicitly incorporated for neural networks [107]. Furthermore, the individual neurons perform local approximation within a region of the domain of the underlying function, without any regard to the data lying outside the combined regions of the individual neurons. For example, neural network with only three nodes in the hidden layer can also be trained to learn the triangular wave function when $x \in [0, 1]$ as shown in Figure 4.3 by excluding the last term in the expression in Equation (4.7). However, this network would perform poorly outside the domain $x \in [0, 1]$. As a result, neural networks can in general, produce unreliable extrapolations of the underlying functional transformation. In fact, the dependence on data can sometimes also result in unreliable interpolation if the training data is not sampled at appropriate granularity. In the following, we make use of the local approximation performed by each neuron to study the impact of width and depth of a neural network.

4.2.2 Wider vs Deeper Networks: Complexity of Local Approximations

The impact of width of a network on its ability to learn different functions can readily be observed from the examples considered in the earlier section. Specifically, we observed that each neuron in the network performs a local approximation of the underlying relationship between x and y . This resulted in the absolute function being learnt with only two nodes while the triangular wave cycle with four nodes by dividing the domain of the training data into regions and learning the function locally in each region with one neuron - in a piece-wise manner. It then stands to reason that a neural network with an arbitrarily large number of nodes and only one hidden layer can, in principle, learn any function with arbitrary precision. However, whether such a network can be trained to identify the values of parameters to learn any function is a question that remains to be addressed. Increasing the width of a network, however provides the network with the ability to divide the domain of the training data to regions with finer granularity thereby allowing it to identify a better representation of the underlying relationship between the data. It must however be noted here that since the activation function of networks is fixed, increasing the width results in more of those nonlinear functional transformations performed by the network. On the other hand, a deep network performs the nonlinear operations in a recursive manner, effectively increasing the nonlinearity of the overall operation. This is because of the fact that while the first hidden layer performs n_1 local operations in the domain of the input data, the second hidden layer performs n_2 local operations in the domain of the output of the first layer, which translates to a more nonlinear operation in the domain of the input data. Figures 4.4 and 4.5 show the activation of individual nodes in a wide network ($n = 5, H = 1$) and a deep network ($n = 2, H = 3$) with *ReLU* and *tanh* as activation functions and randomly generated parameters. It can be observed from Figure 4.4 that the activations of all the three nodes in the networks exhibit similar degrees of non-linearity with different parameters that determine their position and shape. However, Figure 4.5 reveals that the activation of neurons in different layers exhibit different degrees of nonlinearity. Specifically, nodes in the

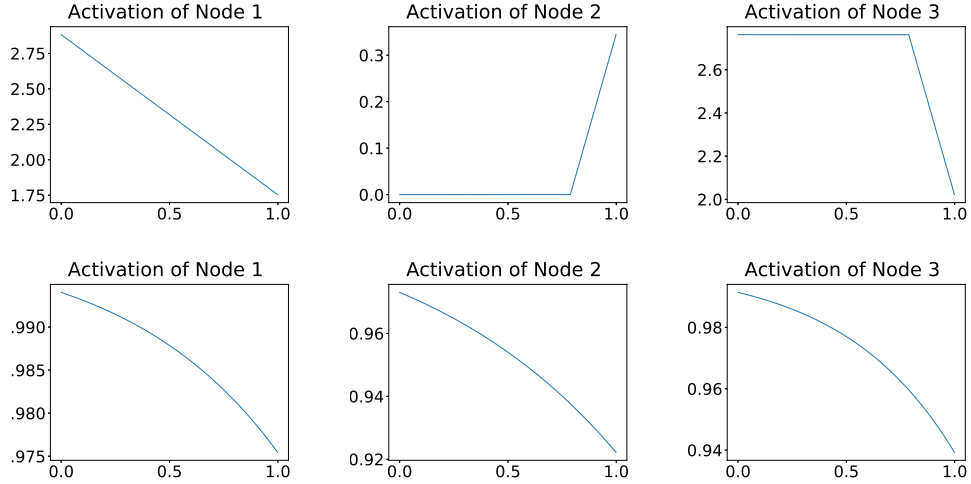
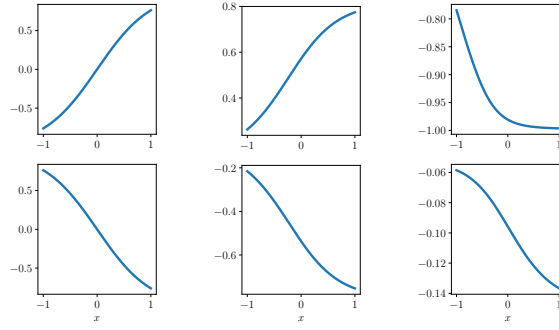


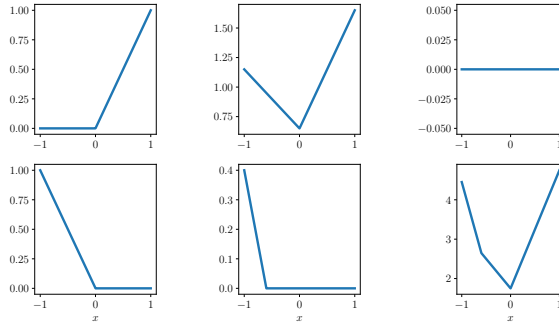
Figure 4.4: Activation of individual neurons in a network with *ReLU* (first row) and *tanh* (second row) as the activation function and three neurons in the hidden layer

first layer exhibit the least nonlinearity, while the nodes in the third layer exhibit the most nonlinearity obtained as a composition of the activation functions.

Therefore, while both width and depth of a network influence the performance of a network, they do so in different manners. Specifically, increasing the width of a network results in increase in the number of local regions of the domain of the function that allows for a finer approximation. However, the extent of nonlinearity of each neuron is the same for such a network. On the other hand, increasing the depth of a network results in recursive composition of the activation function, resulting in increased nonlinearity of the network as a whole. As a result, a user is often provided with the choice of increasing the width and/or depth of a network to improve the performance. However, the above discussion suggests that if one expects highly nonlinear and complex functional forms in the underlying process, it would be a better choice to build a deep network that inherently performs operations of increasing nonlinearity and might be well equipped to approximate the underlying function. On the other hand, a relatively simple function can easily be approximated with a wide network with an appropriate number of nodes in the hidden layers. It must be noted here that the above discussion assumes that it is possible to faithfully train a network such that it identifies the solution to the regression problem. However, challenges such as vanishing



(a) \tanh Layer 1 (b) \tanh Layer 2 (c) \tanh Layer 3



(d) $Relu$ Layer 1 (e) $Relu$ Layer 2 (f) $Relu$ Layer 3

Figure 4.5: Activation of individual neurons in a deep network with \tanh and $ReLU$ as the activation function and 3 hidden layers

gradients and dying units can hinder the learning process, influencing the performance of the network. In addition to these challenges, the degeneracy of solutions in the parameter space also poses threats such as getting stuck in local optima, some of which are described in the next section.

4.2.3 Degeneracy of Parameters

In order to study the degeneracy of parameters in a neural network, let us revisit the problem of learning the absolute value function with the tiny neural network shown in Figure 4.2. While the parameters shown in Figure 4.2 result in the absolute function, an untrained neural network with such a configuration would express the output as a function of the inputs

and parameters as:

$$\hat{y} = \left[w_1^{[2]} \text{ReLU}(w_1^{[1]}x + b_1^{[1]}) \right] + \left[w_2^{[2]} \text{ReLU}(w_2^{[1]}x + b_2^{[1]}) \right] + b_1^{[2]} \quad (4.8)$$

where $w_i^{[j]}$ and $b_i^{[j]}$ represent the i^{th} weight and bias in the j^{th} layer respectively, as shown in Figure 4.6. It can be observed that while Equation (4.6) represents the ideal/desired configuration of parameters, the problem of learning the function from data is seen by an untrained network as described in Equation (4.8) with many more parameters than required to express the function. A careful examination of Equation (4.8) reveals that any combination of the parameters that satisfies:

$$\begin{aligned} w_1^{[2]} w_1^{[1]} &= 1 \\ w_2^{[2]} w_2^{[1]} &= -1, \quad w_2^{[1]} < 0 \\ b_i^{[j]} &= 0 \quad \forall i, j \end{aligned}$$

will result in an exact representation of the absolute function. This indicates the presence of infinite global optima for a network that is as small as shown in Figure 4.6, as opposed to two optima shown in Figure 4.2.

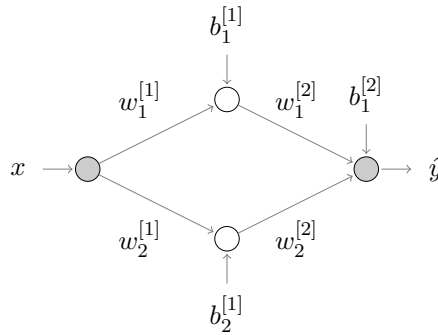


Figure 4.6: Representation of absolute function by an untrained neural network

It can be inferred from the above discussion that as a result of posing a regression problem in the formulation of a neural network, one inherently adds degeneracy to the solutions of

the problem, suggesting that one is highly likely to identify a solution. This is contrary to empirical observations made in training neural networks, where one rarely finds themselves in a global optimum, i.e., with a loss that is equal to zero or the variance of noise. In order to further investigate this contradiction, we study the loss as a function of the weights in a neural network. To that end, we reduce the formulation in Equation (4.8) to one containing only two parameters as shown in Figure 4.7, that can be obtained from Figure 4.6 making the following assignments:

$$\begin{aligned} w_1^{[1]} &= w_1; & w_2^{[1]} &= w_2 \\ w_1^{[2]} &= 1; & w_2^{[2]} &= 1 \\ b_i^{[j]} &= 0, & \forall i, j \end{aligned}$$

The loss function (mean squared error) of this network can then be expressed as:

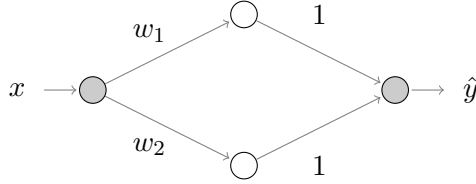


Figure 4.7: Reduced neural network for approximating the absolute function with two parameters

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 = \frac{1}{N} \sum_{i=1}^N (y_i - (\text{ReLU}(w_1 x_i) + \text{ReLU}(w_2 x_i)))^2 \quad (4.9)$$

In order to study the behavior of the above loss as a function of the parameters, we consider $x \in [-1, 1]$ and generate data according to Equation (4.5) and calculate the loss as a function of the weights w_1 and w_2 . Figure 4.8 shows the contour plot of the loss as a function of the weights, from which it can be observed that there are two global optima at $(w_1, w_2) = (\pm 1, \mp 1)$. However, it can also be observed that there are two regions in the $w_1 - w_2$ space the gradient of loss (change in colour in Figure 4.8) is very low. A careful examination of this surface

reveals that there exist line segments described by:

$$w_1 = \pm 1 - w_2; \quad w_1, w_2 \in [-1, 1]$$

where the gradient of the loss with respect to both the weights is equal to zero. These lines represent sub-optimal regions (which can in general be local minima or saddle points) in the $w_1 - w_2$ space. In order to highlight the impact of these sub-optimal regions, we train the reduced network shown in Figure 4.7 with two different initializations: $(w_1, w_2) = (1.7, 0.9)$ and $(w_1, w_2) = (1.9, 0.9)$. Figure 4.9 shows the trajectory of the trained weights for the two initializations where the green and red circles represent the initial and trained weights respectively. It can be observed from Figure 4.9 that small changes in initial conditions of the weights result in distinctly different values of the weights after training. Specifically, while the initialisation $(w_1, w_2) = (1.9, 0.9)$ converges to the global optimum of $(1, -1)$, the initialisation $(w_1, w_2) = (1.7, 0.9)$ encounters a local optimum. We iterated the training of weights for different initial conditions in the range $w_1, w_2 \in [-2, 2]$ with a step size of 0.008 in both directions and identified the set of initial conditions that resulted in convergence to the global optimum. Figure 4.10 shows the set of initial weights that result in convergence to a global optimum (green) and the set that converges to the sub-optimal region (red). It can be observed from Figure 4.10 that for the reduced network with only two parameters, there exists a significant region in the weight space that result in the network converging to only a local optimum, and unable to identify the global solution. This problem of converging to a local optimum can get worse with increase in number of parameters.

Let us revisit the architecture in Figure 4.2. Instead now we set,

$$\begin{aligned} w_1^{[1]} &= 1; \quad w_2^{[1]} = -1 \\ w_1^{[2]} &= w_1; \quad w_2^{[2]} = w_2 \\ b_i^{[j]} &= 0, \quad \forall i, j \end{aligned}$$

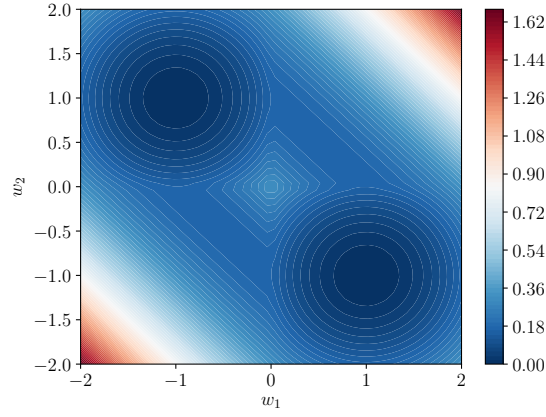


Figure 4.8: Loss of the reduced network as a function of weights

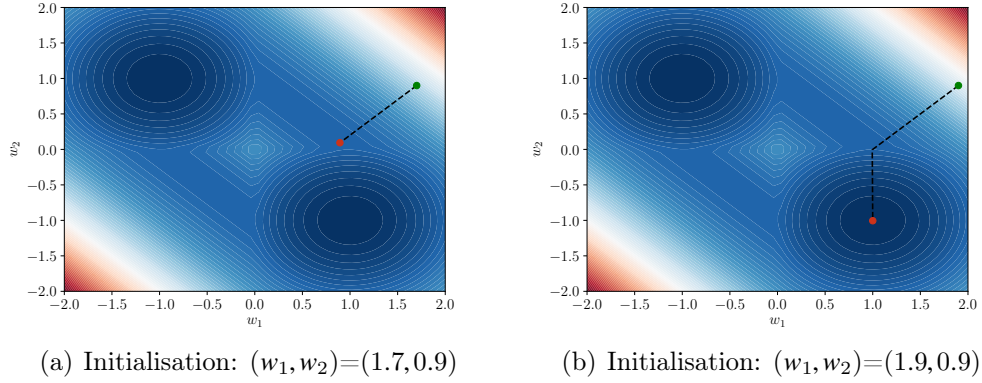


Figure 4.9: Trajectory of weights after training with different initializations (green and red circles represent initial weights and trained weights respectively)

The loss function in this neural network would be given by:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 = \frac{1}{N} \sum_{i=1}^N (y_i - (w_1 \text{ReLU}(x_i) + w_2 \text{ReLU}(-x_i)))^2 \quad (4.10)$$

In this problem, we can see that the loss function is convex in w_1, w_2 . This is owing to the fact that in regression tasks one often uses linear activations in the final layer. Even on inclusion of biases in the final layer, the loss function will be convex in the final layer parameters. The loss function landscape is shown for the loss in Equation 4.10, in Figure 4.11.

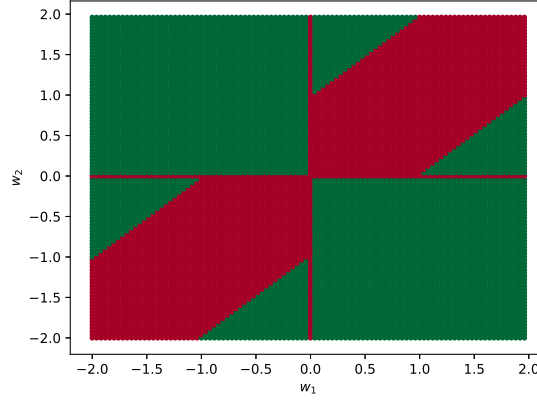


Figure 4.10: Initializations of the reduced network that converge to a global optimum (green) and to a local optimum (red)

So far we have only considered parameters in a single layer. Consider the function $\hat{y} = w_1 \text{ReLU}(w_2 x)$. We can see that the loss function in this case will be given by:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 = \frac{1}{N} \sum_{i=1}^N (y_i - w_1 \text{ReLU}(w_2 x_i))^2 \quad (4.11)$$

One would not expect this loss function to be convex in the parameter space (owing to the nonlinearity). In fact this loss function gives a hyperbolic structure in the parameter space (Figure 4.12a). Training such parameters can be difficult using conventional gradient descent schemes, and may result in suboptimal training. Stochasticity in the gradients can allow us to escape suboptimal regions, however, convergence to a global optimum will be very slow. Now, let us consider the following loss function,

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 = \frac{1}{N} \sum_{i=1}^N (y_i - w_1 \text{ReLU}(w_2 x_i))^2 + \lambda(w_1^2 + w_2^2) \quad (4.12)$$

The loss function in Equation 4.12, penalizes higher values of the parameters. This technique is called regularization, and often leads to smoother function relationships which avoids overfitting. It can be seen that using an L2 norm regularizer (which adds penalty to corresponding to square of the weights as in Equation 4.12), the loss function becomes convex

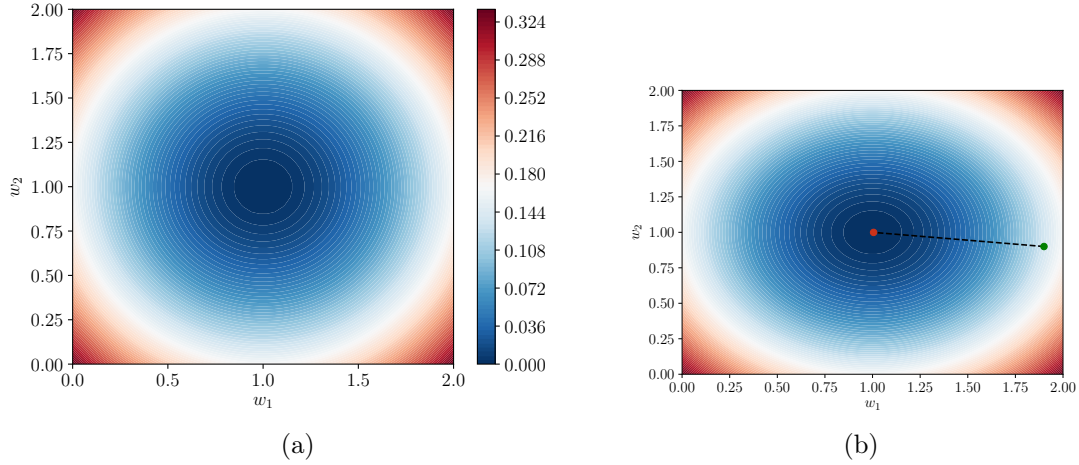


Figure 4.11: (a) Loss of the reduced network corresponding to Equation 4.10 as a function of weights (b) The final layer loss will be uniquely estimated given all the other layers weights and biases

locally in the parameter space (Figure 4.12). This ensures that in the case of training using gradient-based techniques with regularization, one can get optimal training (with bias). In addition to norm-based regularization, dropout is a popularly used technique for regularizing deep neural networks. However, since dropout involves a random dropping of nodes during each forward pass of the input, the impact of such an approach on the loss function is difficult to track and is not presented in this article.

It must be noted that the above discussion presents the final internal representations of neural networks and how they contribute towards achieving a regression task without any regard to the manner in which they are trained. However, the literature of deep neural networks is filled with several advanced techniques for efficient training of the model. The impact of several of these techniques on the internal representations can in fact be easily derived from the above discussion. For example, transfer learning is a technique commonly used to exploit the parameters of a pre-trained model and achieve faster convergence for a related task. In doing so, one starts from the internal representations of the pre-trained model and trains the weights so as to achieve the related task. Such an approach results in the trained representation being adjusted to the new task, and requires less effort compared

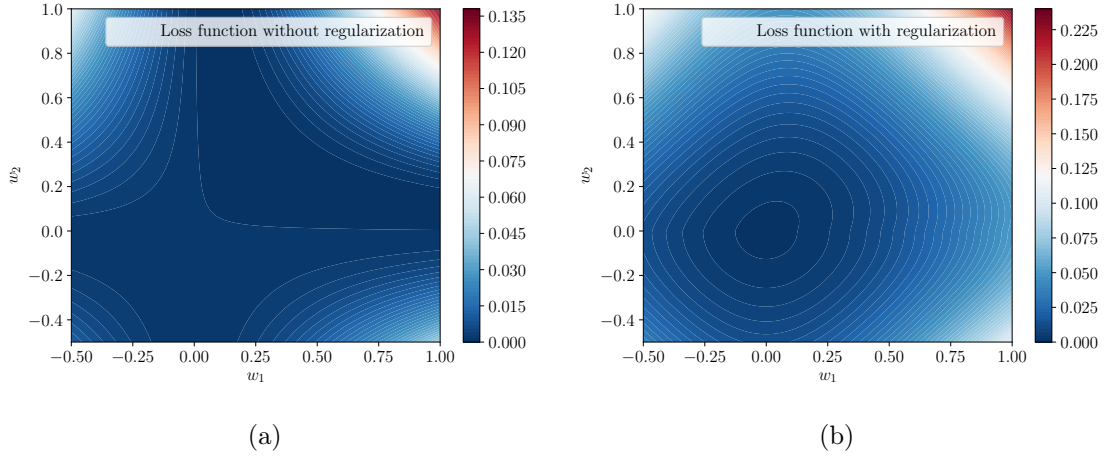


Figure 4.12: Loss function corresponding to Equation 4.11. It can be seen that regularization makes the function locally convex, and helps the training.

to a random representation corresponding to randomly initialized weights. Different optimization techniques that are variants of the gradient descent approach simply work towards efficient learning of weights, and at times attempt avoiding local optima. The differences in optimization approaches along with the severe degeneracy of parameters identified above suggests that different techniques - and in fact the same technique with different initializations - can indeed result in different representations for the same network architecture and same regression task. However, as discussed earlier, the impact of techniques that involve random operations, such as dropout on the representations cannot be traced with the above approach.

Summarizing this section, we have identified the functionality of each neuron in a network and studied the impact of width and depth on the representations of the hidden neurons with respect to the input space. We have also studied the implication of formulating a regression problem in a neural network framework on the degeneracy of solutions in the parameter space. Regularization can help with making the loss function convex with respect to the parameter space (and less flat!). In the following, we present how the individual contributions of neurons are combined to achieve a complicated regression task.

4.3 How does a Neural Network Approximate a Function?: From Parts to Whole

A neural network achieves a regression task by making use of its ability to (i) divide the domain of the training data into finer regions by increasing the number of nodes, (ii) perform increasingly nonlinear operations in each region by increasing the number of layers and (iii) perform different types of nonlinear functions by choosing different activation functions. In the following, we present a detailed discussion on these components with an illustrative example and demonstrate these ideas with the regression of a complicated function.

4.3.1 Illustrative Example: Sinusoidal Function

Let us consider the problem of learning a sinusoidal function as discussed in the previous section. We consider a fully connected neural network \mathcal{N} with L hidden layers that has n_i nodes in the i^{th} layer, and an activation function $g(\cdot)$. We consider the rectified linear unit (*ReLU*), sigmoid (σ) and hyperbolic tangent (*tanh*) activation functions with networks that have 2, 5 or 10 hidden layers, each with 2, 5 or 10 nodes. As earlier, the activation of the final layer was kept as linear in all the configurations. We then trained the networks with RMSPROP optimizer for 500 epochs with a batch size of 128 samples.

The testing data and predictions of the neural networks with *ReLU* as the activation function are shown in Figure 4.13. It can be observed from Figure 4.13 that for the networks with two nodes in each layer, the performance does not improve by increasing the number of layers while when the network has five nodes in each layer, there is a significant improvement in performance compared to their two-node counterpart networks, which also increases with addition of more layers to the network. Finally, when the networks have ten nodes in each layer, the performance is only comparable with the networks with five nodes in each layer, which also does not show marked increase with addition of more layers.

Figures 4.14 and 4.15 present the testing and predicted outputs for all the network config-

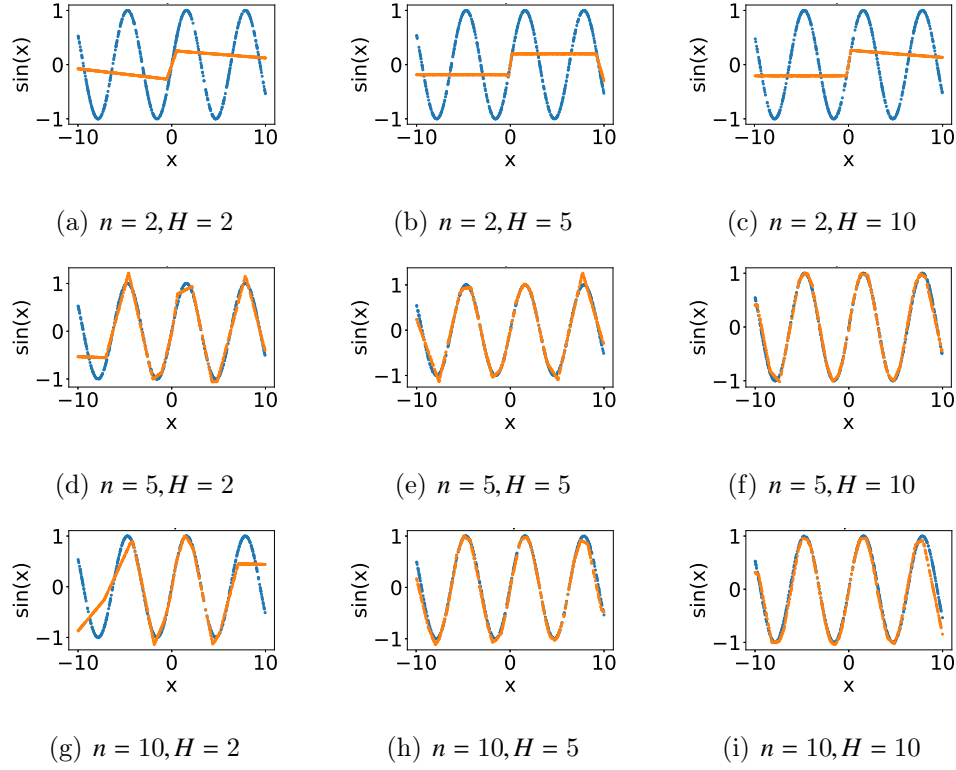


Figure 4.13: Testing data (blue) and model predictions (orange) for different network configurations for approximating the sinusoidal function with rectified linear unit (*ReLU*) as the activation function (n and L represent the number of nodes in each hidden layer and number of hidden layers respectively)

urations with the logistic and *tanh* activations respectively. It can be observed from Figure 4.14 that irrespective of the configurations, the network is unable to adequately approximate the sinusoidal function. On the other hand, Figure 4.15 reveals that for the networks with two nodes in each hidden layer, the performance of the model does not improve by making the network deeper while the networks with five nodes in each hidden layer exhibit significantly better performance than the corresponding models with two nodes. In addition, the performance of these models also increases with addition of more layers to the network. The networks with ten nodes in each hidden layer exhibit a marginal improvement in performance with respect to the networks with five nodes in hidden layers. This performance is also seen to increase with addition of more layers to the network.

A comparison of the performance of models with different configurations and activation

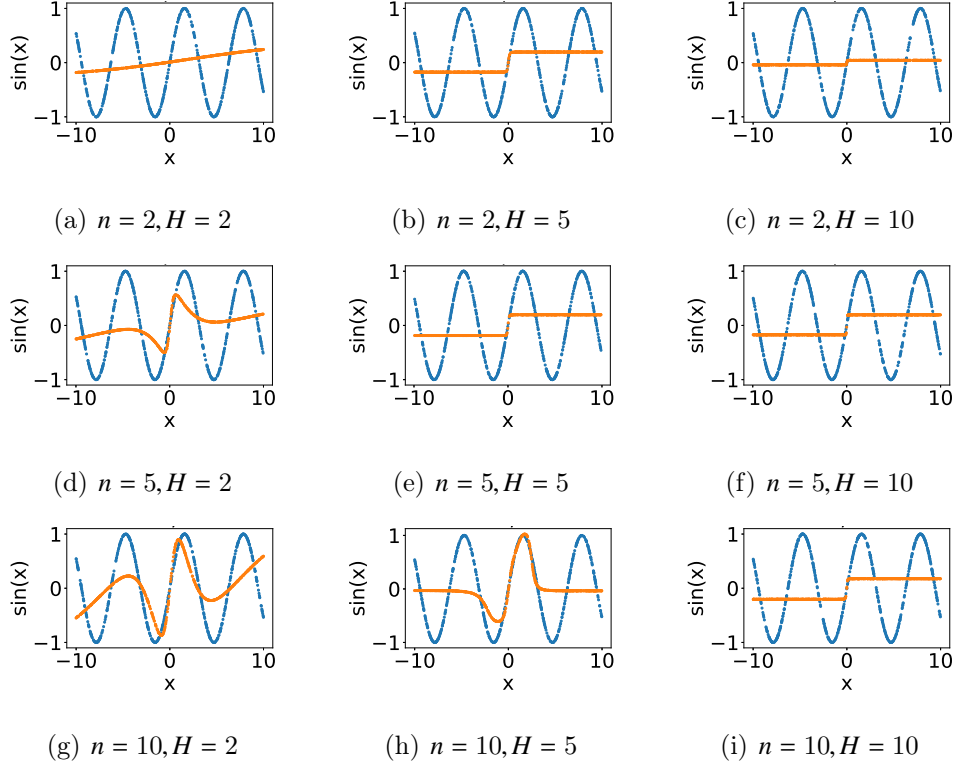


Figure 4.14: Testing data (blue) and model predictions (orange) for different network configurations for approximating the sinusoidal function with sigmoid (σ) as the activation function (n and L represent the number of nodes in each hidden layer and number of hidden layers respectively)

functions, for the present case, reveals that while networks with *ReLU* and *tanh* as activation function are capable of approximating the sinusoidal function, the ones with sigmoid as the activation function are incapable of achieving comparable performance with the same resources. It can also be observed from Figures 4.13 and 4.15 that:

1. As the network is made deeper, the performance improves only when there are at least a certain number of nodes in each layer. Furthermore, the improvement in performance with addition of more layers saturates after a certain threshold.
2. As the network is made wider, the performance improves even when there are very few (2 in this case) number of layers. This improvement in performance also saturates after a certain threshold.

In order to identify the source of this behaviour and further validate the observations

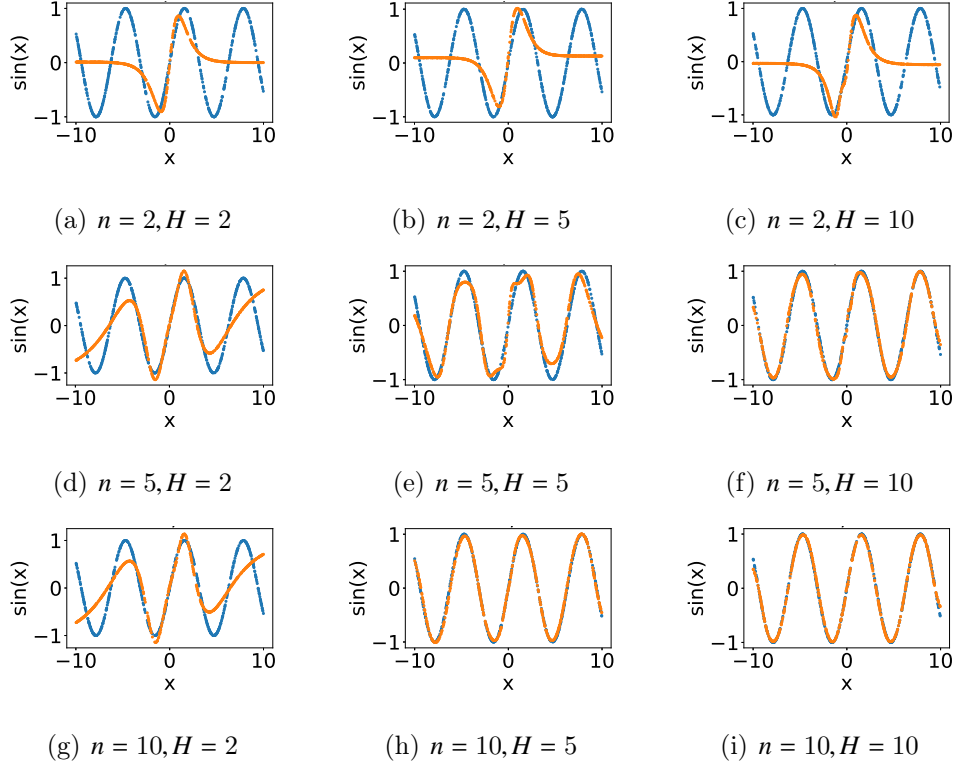


Figure 4.15: Testing data (blue) and model predictions (orange) for different network configurations for approximating the sinusoidal function with hyperbolic tangent (\tanh) as the activation function (n and L represent the number of nodes in each hidden layer and number of hidden layers respectively)

made in the previous section, we study the outputs all nodes in all layers for a few configurations of the networks. Figures 4.16, 4.17 and 4.18 show the activations (also referred to as features) of individual nodes for networks with 5 hidden layers and 5 nodes in each layer with $ReLU$, σ and \tanh as the activation function respectively.

It can be observed from Figure 4.16 that the neurons in the first hidden layer exhibit simple activations, which are then converted into complex structures with addition of more layers, with extremely nonlinear functions obtained in the final layer. On the other hand, the features of the network with logistic function as the activation function do not evolve into complex functions with increased nonlinearity with addition of more layers. The network with \tanh as the activation, however exhibits a similar behaviour as that of the network with $ReLU$ as the activation function, as shown in Figure 4.18. These observations are in line

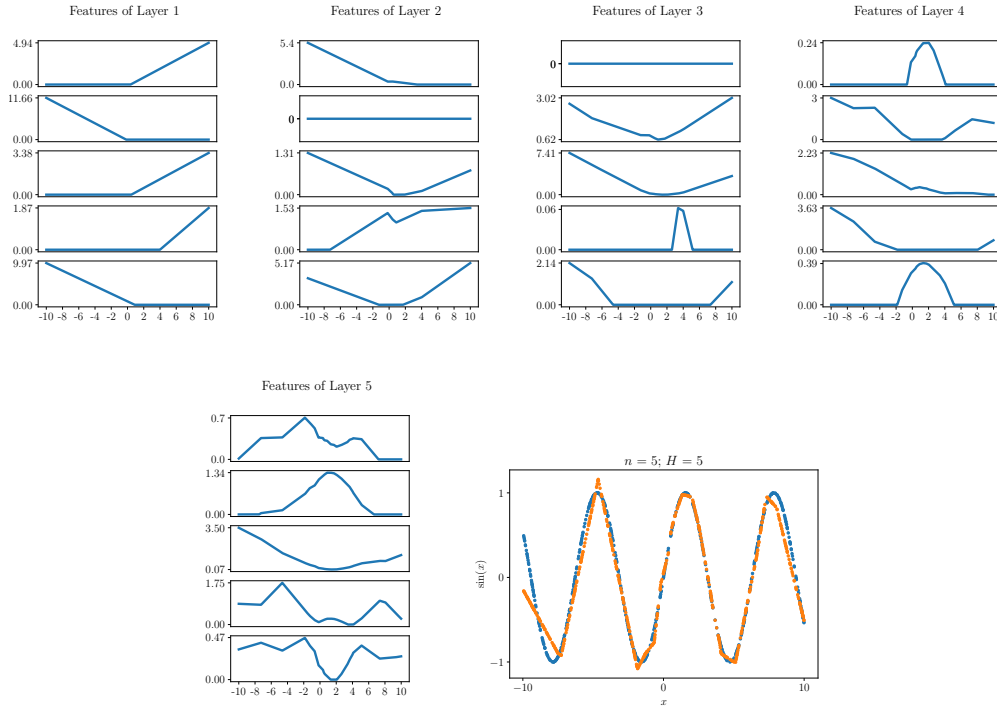


Figure 4.16: Features of hidden layers and final output of the network approximating the sinusoidal function with 5 hidden layers and 5 nodes per layer with *ReLU* as the activation function

with the inferences on the effect of width and depth on the performance of a neural network drawn in the previous section. It is particularly interesting to note how the activations of the network with *tanh* as the activation function result in an increasingly nonlinear function that exhibit sinusoid-like behaviour in the fifth hidden layer of the network. It must be noted here that although the logistic function exhibits a similar shape to the *tanh* activation function, it also exhibits a lower gradient throughout the domain of x which results in slower training and at times poor performance

It is noteworthy that the above results are obtained when the network was trained with RMSPROP to minimize the mean squared error between predictions and the data, and the observations can vary with changes in either of these factors. It is also important to note that while the performance of the network seems to increase with increasing depth/width, this is generally not the case (as will be demonstrated later) and in general results in over-fit models. While increasing depth/width of the neural network increases the degeneracy in the

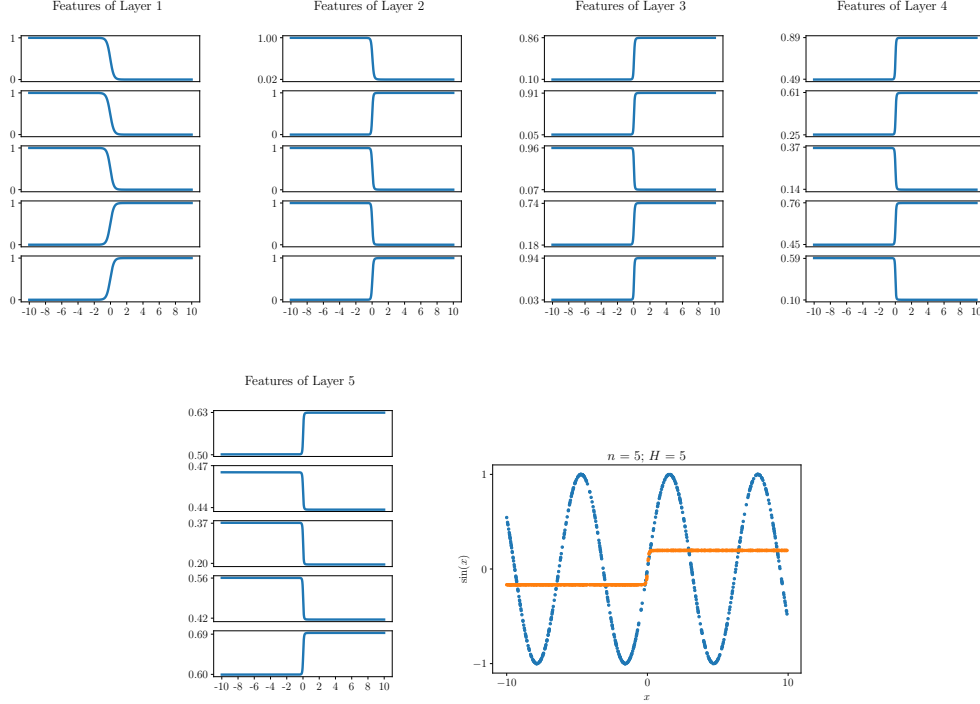


Figure 4.17: Features of hidden layers and final output of the network approximating the sinusoidal function with 5 hidden layers and 5 nodes per layer with σ as the activation function

parameters and results in the network explaining the data well, this comes at the cost of poor extrapolation of the model. It is hence, advisable that one use regularization, in the form of added penalties to higher weights/ drop-out [108].

4.4 Demonstrative Example: Energy Function Landscape

We now present a demonstrative example that involves the approximation of a complicated energy function landscape. These landscapes are often seen in material sciences towards understanding globally stable configurations of crystalline samples, protein folding, structure property relationships, and so on. The data is often in the form of different structures and energy samples, and one attempts to find out the relationships between structure and energy [91], [90]. Descriptors are created from the structure of the molecules, fed as an input to the neural network. The output energy data is often estimated from complex

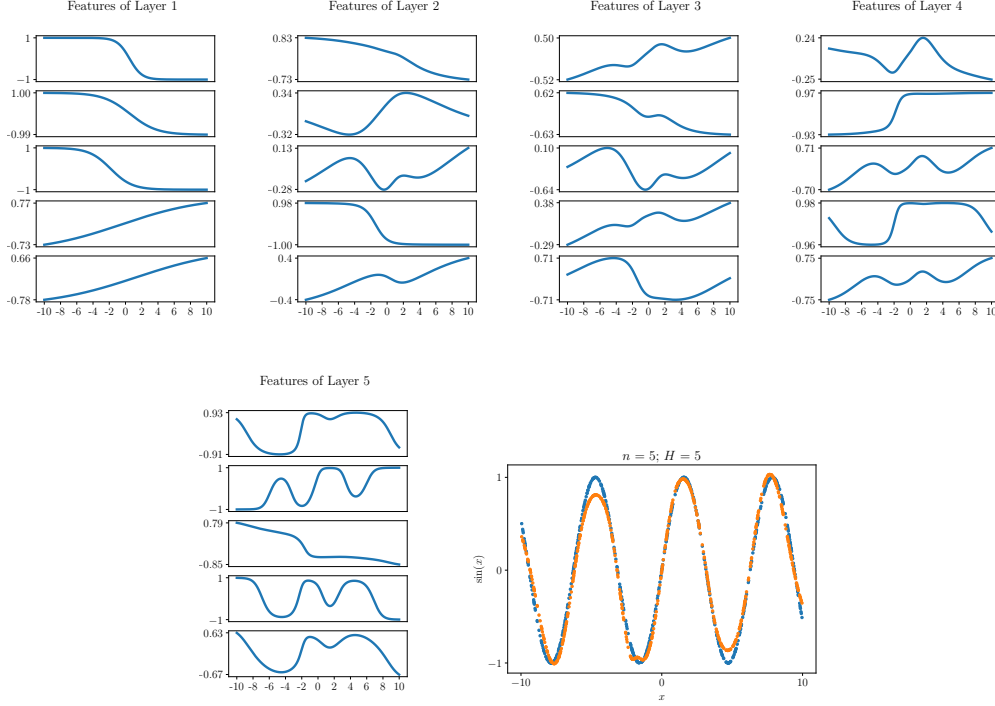


Figure 4.18: Features of hidden layers and final output of the network approximating the sinusoidal function with 5 hidden layers and 5 nodes per layer with *tanh* as the activation function

energy functionals using Density Functional Theory calculations.

One representative model of these energy function landscapes, is the shekel function. This is a multi-modal function in an arbitrary number of dimensions that can be expressed as:

$$y = - \sum_{i=1}^m \frac{1}{c_i + \sum_{j=1}^n (x_j - a_{ji})^2} \quad (4.13)$$

The equation is characterized by the number of modes m , and the dimensionality of \mathbf{x} , n . For the demonstrative example we choose 17 modes ($m = 17$), in a 2 dimensional space $n = 2$. The parameter values are listed in Table 4.1. The domain of the function lies in bound \mathbb{R}^2 , within $[-5, 5]$. A major characteristic of the shekel function is that c_i represents the strength of a mode, and $\mathbf{a}_i = [a_{1i}, a_{2i}]$ represents the location of the mode. The interaction between individual modes functional form results in the final input output relation as shown in Figure 4.19a. This can also be visualized as a mapping from the 2D input to the 1D output Figure

4.19b.

Table 4.1: Parameters of the shekel function for the demonstrative example

Mode (i)	c_i	a_{1i}	a_{2i}
1	1.7	-1	-1
2	1.53	0	-1
3	1.87	1	-1
4	2.55	1	0
5	1.19	1	1
6	1.36	0	1
7	2.04	-1	1
8	1.7	-1	0
9	0.85	0	0
10	1.7	-2	-2
11	1.53	0	-2
12	1.87	2	-2
13	2.55	2	0
14	1.19	2	2
15	1.36	0	2
16	2.04	-2	2
17	1.7	-2	0

Usually these landscapes exhibit multiple local minima in the configuration space, and hence are difficult to model analytically. A common technique used in such cases is to identify descriptors of the different configurations, and create a descriptor-energy model in a supervised manner. Neural networks are increasingly useful in this regard. This is however contingent on the energy estimates from Density Functional Theory calculations. There are however, other concerns with the training of these neural networks.

In recreating the energy landscape from data, our goal is to approximate the energy landscape as well as possible, and identifying a minimal set of network hyperparameters (width and depth). For this we focus on using one activation function – *tanh*, as it is seen to be performing well for smooth function approximation task. The neural network is trained with sufficient samples from the domain, densely sampled from the regions of minima. This ensures that any underfit during the training exercise is an artefact of the training, and not dearth of samples. Each network is trained for 500 epochs.

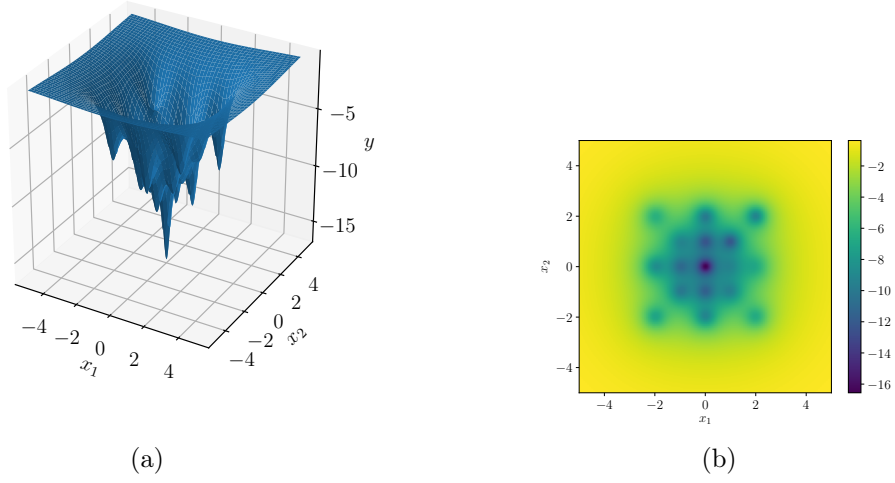


Figure 4.19: Shekel function described using the parameters in Table 4.1(a) 3D representation of the function approximation task (b) Shows final mapping of the 2D input to the 1D output

4.4.1 Effect of Depth and Width on the Predicted Energy Landscape

Here, a depth-width study is done towards understanding neural network performance according to Table 4.2. We start with a study on the effect of depth a neural network, by considering 10 nodes in each hidden layer.

Table 4.2: **Depth-Width Effect:** Combinations of n and H considered for training shekel function

n/H	$H = 3$	$H = 10$	$H = 20$
$n = 5$	✓		
$n = 10$	✓	✓	✓
$n = 50$	✓		

As seen in Figure 4.20a, the network with $n = 10$, $H = 3$, finds the overall feature of the model (location of the global minima). The issue however is that with this shallow architecture with just 3 hidden layers, the granular features of the landscape are not predicted by the network. An obvious choice in the meanwhile, is to increase the depth. Increasing the depth of the neural network initially shows that it learns the more granular features, however, depth of a neural network does not significantly increase the performance of the

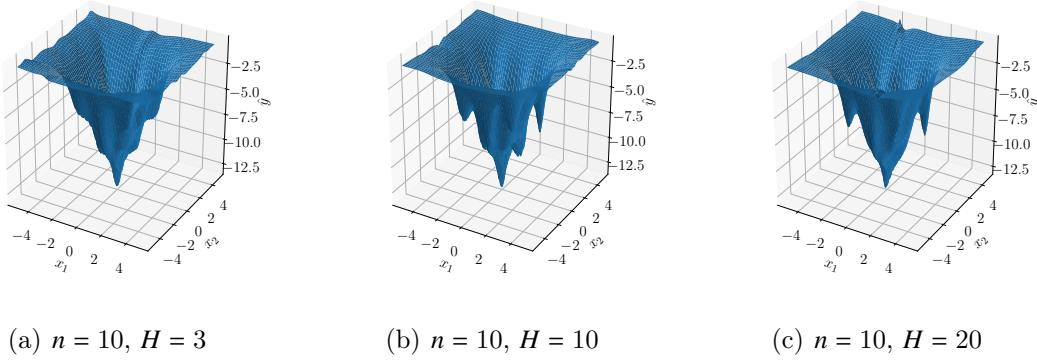


Figure 4.20: Increasing depth (H hidden layers) does not yield increase in performance for a network with 10 nodes per hidden layer

neural network despite a width of 10 nodes in each layer. This is however expected as increasing the depth of the network results in formation of multiple suboptimal regions in the loss landscape, as seen in earlier section. Though we equip the network to be able to approximate a wide range of functional forms, accessibility of these varied functional forms through the parameter space is a huge problem for the network. Depth increases model complexity, hence is difficult to train.

In the meanwhile, increasing the width of the neural network, shows better function approximation (Figure 4.21). The depth is kept constant at $H = 3$. Network with width of 5 nodes captures the broad detail of the landscape, increasing the width shows better performance, but to estimate the finer features, we have to use a network with 50 nodes. In the previous section we showed that the loss function is always convex with respect to the final layer parameters, irrespective of the width of the final layer. While increasing the width does give rise to degenerate solutions, it is still easier to train than deep networks. To get better insight into what the network does in each of the cases, we need to look at what each nodes mapping would be post training.

4.4.2 Node specific local approximations

The shekel function has a final output activation as shown in Figure 4.19b. This idea can be extended to visualizing the value of the hidden nodes. In this section we compare two

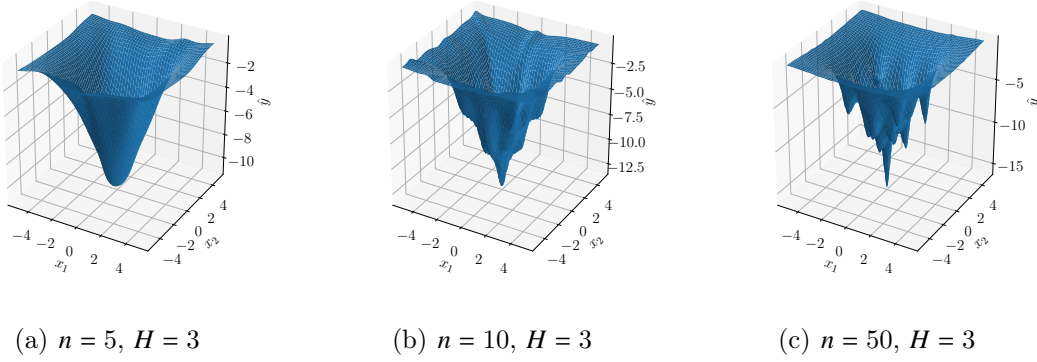


Figure 4.21: Increasing width (n nodes per layer) of the neural network increases the approximation accuracy of the network with 3 hidden layers

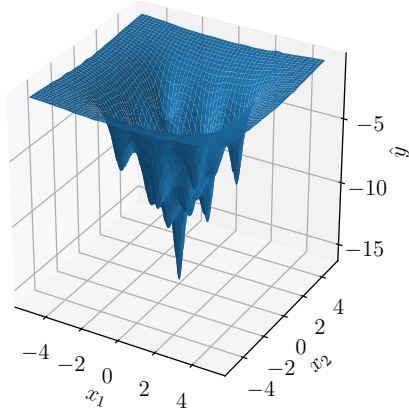
networks: one network is wide and shallow (\mathcal{N}_1 : $n = 50, H = 3$), and the other is thinner and deeper (\mathcal{N}_2 : $n = 20, H = 5$). From Figure 4.22, it can be seen that these two networks have similar performance.

$$\mathcal{N}_1 : \quad n = 50, \quad H = 3$$

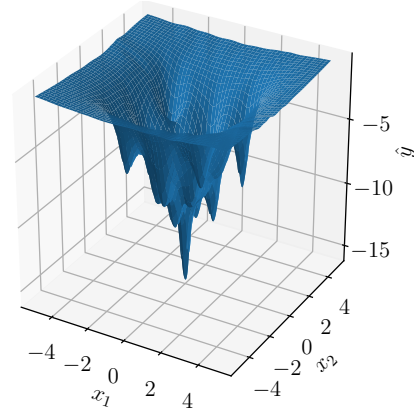
$$\mathcal{N}_2 : \quad n = 20, \quad H = 5$$

Figures 4.23, 4.24 show that as we go deeper into the network, more complex patterns are generated as we go deeper into the neural network. This is owing to the fact that deeper networks results in more nonlinear operations on the features of the previous layer, leading to nonlinear patterns. In fact if we look at the final hidden representations of the two networks (Figure 4.25), we see that the deeper network, \mathcal{N}_2 has more complex features than the shallower network, \mathcal{N}_1 .

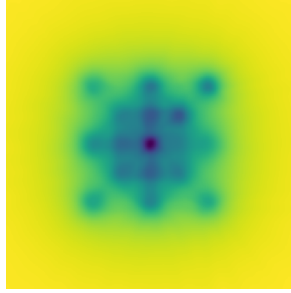
Comparing similar performing networks, tells us that one could have similar performance by increasing depth or increasing width of a neural network. This however, has a couple of caveats associated with it. Increasing depth and width, increases the complexity of the loss function with respect to the parameters. In both these situations, it is difficult to find an optimum set of parameters. However, we note that deepening leads to more difficult training. This could be as a result of having hyperbolic regions of flatness (mentioned in



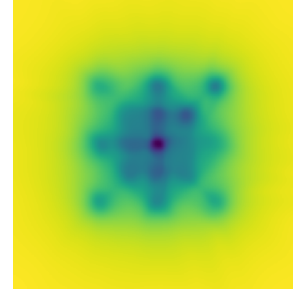
(a) Output: \mathcal{N}_1



(b) Output: \mathcal{N}_2



(c) Input-Output Mapping: \mathcal{N}_1



(d) Input Output Mapping: \mathcal{N}_2

Figure 4.22: Network comparison: \mathcal{N}_1 and \mathcal{N}_2 . Both have similar performance.

earlier section), while in increasing width it is due to degeneracy of the intermediate features. Increasing depth leads to estimating more complicated features (Figure 4.25a), increasing width gives us a wide range of simpler functions (Figure 4.25b).

4.5 Major Results

There is an inherent trade-off between depth and width of neural networks. Similar performance can be obtained by increasing the depth or the width of neural networks. While increasing depth leads to more complex hidden representations deeper into the network due to nonlinear operations, increasing width combines many simpler features into complex ones.

However, both these tasks inherently lead to complicating the loss function landscape

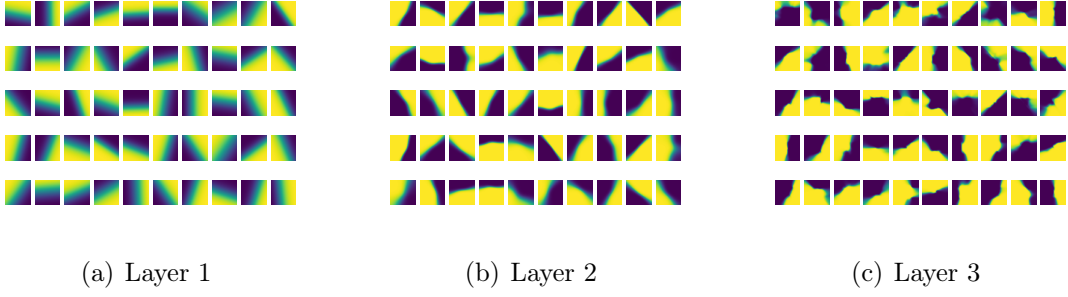


Figure 4.23: Input region activation of each node in each layer for \mathcal{N}_1 . Each layer plot contains activation of each of the node corresponding to the input space. Yellow is a high value of the activation and blue corresponds to low activation (+1 and -1 respectively) since we are using *tanh* activation function for each layer

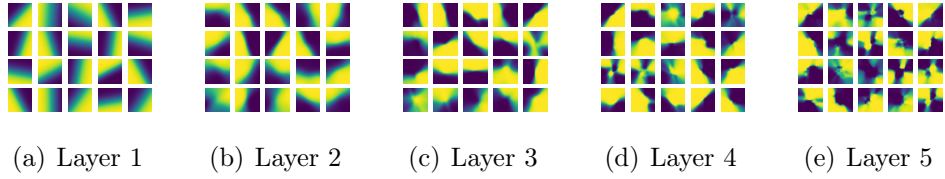
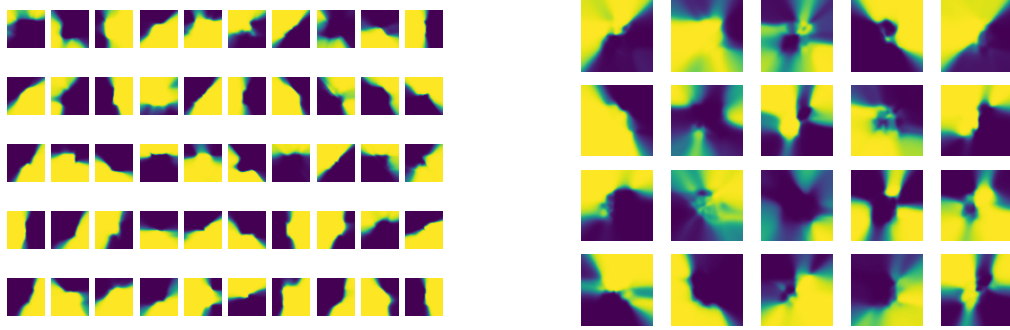


Figure 4.24: Input region activation of each node in each layer for \mathcal{N}_2 . Each layer plot contains activation of each of the node corresponding to the input space. Yellow is a high value of the activation and blue corresponds to low activation (+1 and -1 respectively) since we are using *tanh* activation function for each layer

with respect to the parameters. Multiple local minima and suboptimal regions are formed in the process on increasing the dimensionality of the network. So much so that even with high quality data with no noise, the network sometimes is still unable to perform the function approximation task. Due to the degeneracy in loss function, regularization aids in making the landscape convex with respect to the parameter space. This is also supported by work by [109].

Further, we note that just increasing the depth of the network does not necessarily increase the performance of the network. High depth, low width networks still do not have the necessary ability to map the input to the output with low loss. This was shown in the context of approximating triangular wave functions, sinusoidal functions and approximating the Shekel function, for each of which we need a given width for mapping the input to the



(a) Final Hidden Layer Representation \mathcal{N}_1

(b) Final Hidden Layer Representation \mathcal{N}_2

Figure 4.25: Input region activation of each node in the final layer for \mathcal{N}_1 and \mathcal{N}_2

output, well.

This leads us to one of the more important questions – Is the performance poor because of suboptimal training of the network or due to suboptimal architectures? In the case of learning a complex functional relationship (Shekel function), we noted that similar characteristics are exhibited in both these problems. It is crucial in this regard that width plays an important role. While high width and low depth does not allow for necessary ability in the network to nonlinearly modify the feature space and may result in overfitting, just increasing the depth of the network without considering the width, is not a viable option either. Convexity of the loss function helps in the training exercise, and we note that the loss function is convex with respect to the final layer parameters, given the parameters from every other layer.

In recent years a lot of focus has gone into creating optimal training algorithms for training a neural network [104, 106, 105]. These training strategies are all in essence variants of traditional stochastic gradient descent and it can be argued that for training neural networks using gradient descent schemes, stochasticity is indeed important to navigate really complex loss function landscapes (including smart parameter initializations). This allows the network to escape suboptimal regions in the loss landscape that conventional gradient descent strategies will be unable to do. And then there is of course the concern of overfitting. It must be noted that the results and discussion presented above are based on noise-free data and thus

represent solely the features of neural networks as a machine learning algorithm. However, real life data are almost always corrupted with noise that further introduce complications in training the network. The impact of noise on the performance of the network as a function of the size of the network, especially the issue of overfitting is a very well-known property in machine learning, and is not studied in the article. While we have not addressed overfitting in this article explicitly, we note that regularization is the way to avoid overfitting. Further, regularization creates relatively less complex loss landscapes with respect to the parameters for us to navigate, and can indeed lead to better training of the network. One can avoid overfitting during the training exercise by regularizing (penalizing high values of the parameters, or using techniques such as drop-out [108]).

However, there is little progress in the latter question of optimal architectures for training. We are at present unable to say whether a deep architecture is indeed capable of capturing a functional relationship. To give an analogy, if one tries to fit the functional form $y = |x|$ with a really deep network but of a width of 1 unit, no matter the depth of the network, one cannot capture the relationship between y and x , irrespective of the activation function used. This brings us to a very important unsolved problem, what is the minimum architecture required to solve such a task. At present we use convenient heuristics based on the data distribution, and that is the guiding principle for regression tasks. More complicated distributions as in the shekel function require more parameters (more depth/width), while distributions such as the absolute value function may require lesser parameters. Architectures are highly dependent on the data generating process and one would need custom architectures for specific problems in chemical engineering. For example, it is not ideal to use a simple high depth neural network for functions such as sinusoids and one could potentially do better with a recurrent architecture owing to the periodic distribution form for the function. A lot of recent publications have attempted to answer this question for special class of problems, and it would be interesting to see how these translate to specific problems in the domain of chemical engineering.

Chapter 5: Mechanistic Explanation Generation (MEG)

To address the challenge of explainability posed by neural networks, we present an explainable AI framework for mechanistic explanation generation (XAI-MEG) that combines techniques from machine learning (we call this *numeric* AI), with first-principles-based knowledge that is modeled using *symbolic* AI techniques. Symbolic AI refers broadly to the methods developed in the 1970s and '80s during the *expert systems era* to represent symbolic knowledge and model inference [1].

This chapter is organized as follows: In Section 5.1, we explain the different modules in the AI. This is followed by Section 5.2, where we consider different process case studies and models that often arise in many chemical engineering applications, with the results provided by the XAI.

5.1 AI for Mechanistic Explanation Generation – XAI-MEG

As we know, human modeling experts have the ability to study the data and the process, develop insights about possible underlying mechanisms, and then formulate the model in an iterative process of trial-and-error. They are further able to explain their reasoning and provide causal explanations. They are also often able to judge which models are likely (and which ones are unlikely) for a given system, given the data. The current machine learning systems are nowhere near such capabilities. This is the deficiency we try to address and our work is just an early step in the long journey ahead.

In developing various model hypotheses, there is a dynamic interplay between symbolic interpretations of data and numerical estimation of the features (i.e., the combination of variables) to test the hypotheses. The symbolic interpretation deals with the idea that

certain features mean certain physicochemical interactions, and restricting the features to such meaningful interactions would aid in better explainability. For example, in a dynamical system, the presence of the second-order space derivative term in a model corresponds to diffusion, and the first-order temporal difference is a result of the rate of change at a point in space. For a machine to make similar hypotheses, we need to inform the intelligent system of such a priori fundamental knowledge, and specify the allowed functional transformations and combinations of variables to make mechanistically plausible models.

For example, one often finds elementary functions such as x^2 , e^x , $\ln x$, $\sin x$, and $\cosh x$ in chemical engineering models that can be explained by relating them to first-principles mechanisms of the underlying physics and chemistry. One rarely comes across terms such as $e^{\cosh(x^2)}$, $\ln(\sin^2 x + \ln(x))$, $e^{xe^{xxe^x}}$, etc. Such complicated expressions have often been identified in symbolic regression and in other black-box models [110, 111]. Although such needlessly complex models may fit the data well, they are quite unhelpful in understanding the underlying physics and/or chemistry of the system. There is no physicochemical mechanism that would have generated such functional forms. We gain no mechanistic insights from a model of such needless complexity. They are hardly different from black-box neural network models, which are also known to fit the data well but with no interpretability. In our approach, we try to avoid such a fate by building-in a priori first-principles-based elementary functional forms which are part of the knowledge base (Figure 5.1: Knowledge Base and Feature Extractor). This is part of the *symbolic* AI component in XAI-MEG.

The models created by XAI-MEG involve ordinary and partial differential equations derived using the identified features (Figure 5.1: Model Estimator). This is guided by templates of different spatio-temporal models. For example, in the case of reaction-convection-diffusion systems, this would be analogous to estimating models between rate of change of variables, gradients of the concentrations, and various orders of reaction terms.

The identified model is then sent to the explanation generator (Figure 5.1: Explanation Generator) that also takes inputs about the symbolic variables and the features generated

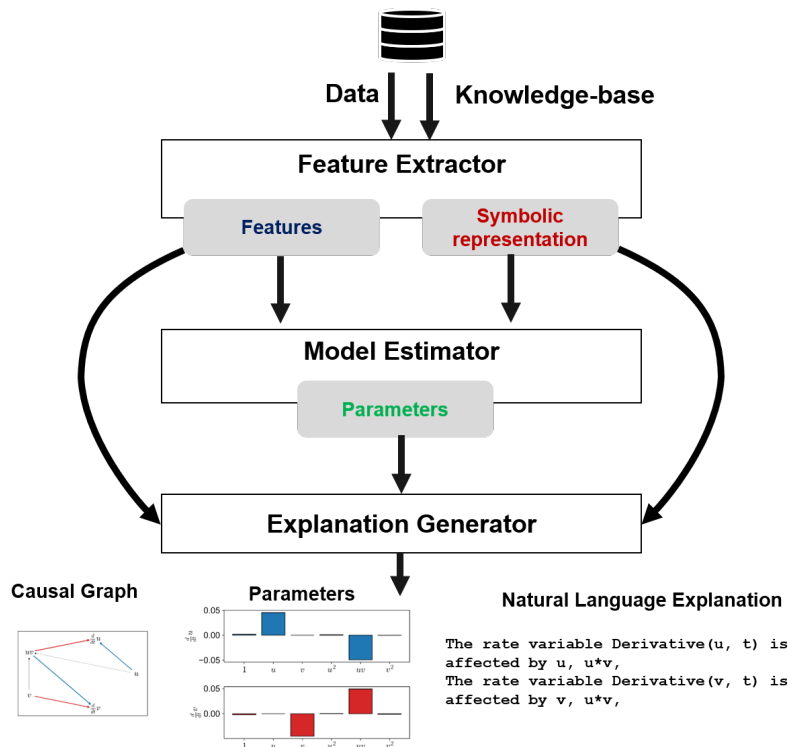


Figure 5.1: Architecture of XAI-MEG

to generate causal models. These are represented as signed directed causal graphs where the directed edges represent the flow of causality and the signs indicate their influence (i.e., influence positively or negatively). These types of causal models are useful to understand which variables and variable transformations in the physicochemical system affect the system dynamics to gain mechanistic insights. Phenomena like *convection*, *diffusion* and *reaction* are defined based on the features extracted from data.

While partial measurements of the hidden state of the system is definitely a concern, we note that \mathbf{u} is observable at several points in space and time. Other issues such as the error in the measurement are a concern as well, but denoising techniques can be used to generate simple models from data [19, 112]. Based on the process model, the unknown reaction/generation term, $\mathbf{R}(\cdot)$, is expressed based on plausible higher-order terms in \mathbf{u} . An additional assumption made in our approach is that different phenomenon contribute additively in a PDE model. The level of contribution depends on the parameters of said

features. Overall, we assume that:

1. Features of the state of the system are meaningful physical variables
2. All relevant variables of the system are observed
3. The system model does not vary with time and space, and the modes of operation of a relevant variable is constant (i.e., parameters Ξ is constant and not spatiotemporally varying)
4. The measurement data has been preprocessed and denoised.

In the following subsections, we delve deeper into the different models used for system identification for explanation, how features are extracted from measurements, method for identification of models, and finally explanation generation using the identified process model.

5.1.1 Temporal and spatio-temporal models for explanation generation

Dynamical systems are often modeled in the form of the state-space evolution of the system. In these models, the state of the system is mapped to its evolution, and measurements are made of these states, such as

$$\begin{aligned}\frac{d\mathbf{u}}{dt} &= \mathbf{f}(\mathbf{u}(t), \bar{\boldsymbol{\beta}}) + \boldsymbol{\eta}(t) \\ \mathbf{y} &= \mathbf{g}(\mathbf{u}) + \boldsymbol{\epsilon}(t)\end{aligned}$$

where $\mathbf{f}(\cdot)$ is a functional field that maps states of the system (\mathbf{u}) to corresponding rate of changes of the state $\left(\frac{d\mathbf{u}}{dt}\right)$, based on parameters $\bar{\boldsymbol{\beta}}$. The measurements \mathbf{y} depend on a nonlinear transformation $\mathbf{g}(\cdot)$ of the states. It is noted that there are also noise variables in the process model $\boldsymbol{\eta}(t)$ and measurement model $\boldsymbol{\epsilon}(t)$, which need to be considered. Many real world problems often come in these forms of nonlinear differential equations. However, many nonlinear systems can be approximated as a nonlinear function of the states of the

system, but linear in parameters, i.e.,

$$\frac{d\mathbf{u}}{dt} = \Theta[\mathbf{u}]\Xi \quad (5.1)$$

$$\mathbf{y} = \mathbf{g}(\mathbf{u}) + \epsilon(t) \quad (5.2)$$

where $\Theta[\mathbf{u}(t)] = \begin{bmatrix} u_1 & u_2 & \cdots & u_1^2 & u_2^2 & \cdots & u_1 u_2 & \cdots & u_{d-1} u_d & \cdots \end{bmatrix}$, is a nonlinear mapping of the states to a higher dimensional vector, and $\Xi = \begin{bmatrix} \xi_1 & \xi_2 & \cdots & \xi_d \end{bmatrix}$ are set of parameter vectors (d is the dimensionality of the state vector). The parameter vector corresponding to the rate variable i , ξ_i , says which modes of operations/features affect the rate variable.

Temporally varying data come in the form of a 2-dimensional data with time and variables. This could be written as $\mathbf{u}(t_j)$, where $\mathbf{u} = \{u_i\}$ is the set of individual variables. For these, systems models can be estimated in the form shown in Equation 5.1. The parameter vectors ξ_i can then be sparsely identified to recreate the observed rates of changes in relevant variables. Meanwhile, in a spatio-temporal system, the state of the system is given for each point in time at a given point in space, and can thus be written in the form of a tensorial data $\mathbf{u}(t_j, \mathbf{x}_k)$, where $\mathbf{u} = \{u_i\}$. This tensorial representation, gives insight about the value of the variable (i), at time (j) and at a point in space (k) given by u_{ijk} .

Most spatio-temporal dynamical models can be seen from a reaction-convection-diffusion standpoint. Consider a system with states $\{u_i\}$, in a velocity vector field \mathbf{v} . In addition to this convective transport of the states, there is also a diffusive transport characterized by diffusivities $\{D_i\}$. The states are generated at a rate $\{R_i(\mathbf{u})\}$. This results in the state-space model for the reaction-convection-diffusion system of the form,

$$\frac{\partial \mathbf{u}}{\partial t} = -\nabla \cdot (\mathbf{v}\mathbf{u}) + \nabla \cdot (\mathbf{D}\nabla \mathbf{u}) + \mathbf{R}(\mathbf{u})$$

Under conditions of constant diffusivity and incompressible flow, this can be rewritten

as,

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{v} \cdot \nabla) \mathbf{u} + \mathbf{D} \nabla^2 \mathbf{u} + \mathbf{R}(\mathbf{u})$$

We see that other phenomenological spatio-temporal models are restrictions to this model form. For example, under no convection, the equation reduces to the standard multicomponent reaction-diffusion systems,

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{D} \nabla^2 \mathbf{u} + \mathbf{R}(\mathbf{u})$$

The dynamics of these systems is a combination of terms corresponding to the temporal gradients, spatial gradients and parameters of the system. The features considered, hence, must include the variable of interest (\mathbf{u}), the temporal difference ($\partial_t \mathbf{u}$), the spatial derivatives ($\nabla_{\mathbf{x}} \mathbf{u}$, $\nabla_{\mathbf{x}}^2 \mathbf{u}$), and higher order terms of \mathbf{u} . In the next subsection we discuss how these features are generated from data.

5.1.2 Feature extraction from measurements

For temporal models, polynomial features are generated from the measurements, resulting in a higher-order feature matrix $\Theta[\mathbf{U}]$. These features include linear (u_i), quadratic ($u_i u_j$) and cubic feature generation ($u_i u_j u_k$). For spatio-temporal systems, in addition to the higher order feature matrix, we also estimate the time and space derivatives in data using finite difference. As mentioned earlier, the important features we need to extract from the system of interest are $\mathbf{u}(t, \mathbf{x})$, $\nabla \mathbf{u}(t, \mathbf{x})$ and $\nabla^2 \mathbf{u}(t, \mathbf{x})$, the gradient terms depend on the spatial resolution. $\partial_t \mathbf{u}$ is estimated based on the sampling-time from the system. Given that the measurements provided are denoised, it is possible to employ finite derivatives in order to create temporal and spatial derivatives directly. As noted earlier, the finite-differencing strategy is contingent on the frequency of the measurements and noise in the data. Denoising strategies and interpolation techniques can be employed to get an accurate estimate of these variables [19],

and their gradients and Laplacian.

The feature generation for the gradient in the rest of the work is done using central differencing,

$$\left. \frac{\partial u}{\partial x} \right|_{loc=i} = \frac{u(x_{i+1}) - u(x_{i-1}))}{2\Delta x}$$

The Laplacian is estimated using the gradient $\partial_x u$ as,

$$\left. \frac{\partial^2 u}{\partial x^2} \right|_{loc=i} = \frac{\partial_x u(x_{i+1}) - \partial_x u(x_{i-1}))}{2\Delta x}$$

For spatio-temporal systems, we customize the regression features $\Theta_i[\mathbf{u}]$ based on the variable of interest u_i . Time derivative is first estimated from the t, u_i data, using finite differencing. The spatial derivatives are only generated for the variable of interest ∇u_i (convection) and $\nabla^2 u_i$ (diffusion). These features are then concatenated with higher order features corresponding to the reaction terms. This would result in models of the form,

$$\begin{bmatrix} | \\ \frac{\partial u_i}{\partial t} \\ | \end{bmatrix} = \begin{bmatrix} | & | & | & | & | & | & | & | & | & | & | \\ \nabla u_i & \nabla^2 u_i & u_1 & u_2 & \cdots & u_1^2 & u_2^2 & \cdots & u_1 u_2 & \cdots & u_{d-1} u_d & \cdots \\ | & | & | & | & | & | & | & | & | & | & | \end{bmatrix} \begin{bmatrix} \xi_i \end{bmatrix} \forall i \in \{1, \dots, d\}$$

5.1.3 Model Estimator

Once the features are generated, the model estimator module takes as inputs the derivative and polynomial features generated by the feature extractor, $(\{\Theta_i[\mathbf{u}]\}, \partial_i \mathbf{u})$, and symbolic representation of the features, to develop a linear parametric model (Equation 5.3).

$$\frac{du_i}{dt} = \Theta_i[\mathbf{u}] \Xi \quad \forall i \quad (5.3)$$

The estimated time derivatives, $\frac{du_i}{dt}(t)$, and features, $\{\Theta_i[\mathbf{u}]\}$, are then used to estimate the parameters for each feature i , based on the optimization problem:

$$\arg \min_{\xi_i} \left\| \frac{du_i}{dt} - \Theta_i[\mathbf{u}] \xi_i \right\|_2^2 + \lambda \|\xi_i\|_1 + \lambda(1 - \alpha) \|\xi_i\|_2^2 \quad \forall i$$

This is an elastic-net optimization problem, where the hyper-parameters are estimated using a λ sweep. A bias term (denoted by symbolic variable ‘1’) is also added to estimate an affine relationship between the outputs and the regressors, instead of a linear relationship. The optimal ξ_i is estimated using the region where the R^2 score starts decreasing, rapidly. Based on the parameter λ , we can promote sparse models for this identification exercise. Hyper-parameter is tuned based on the R^2 score via cross validation. A general cutoff of the smoothened reconstruction with an R^2 of 0.99 is considered for a good/sparse model. While R^2 does weakly increase with number of features, since all models are sparse, a uniform cutoff is chosen. This in-fact is a hyperparameter that should be tuned, and will depend on the problem.

5.1.4 Explanation Generation

In recent years, most natural language generation work involves showing a wide range of examples to an AI system so that it is able to create language models. However, such a large corpus of examples is typically lacking for chemical engineering systems. Therefore, we require a different approach for our task. We require a system that uses symbolic interpretation of features in the model, and automatically tags model features and equations into different classes such as convection, diffusion, reaction, etc. This is similar to a *multi-class classification problem*, where each model form could have multiple phenomenology.

Each identified feature is tagged based on the symbolic manipulation of the main variables. This symbolic manipulation along with the parameters is used to explain the model based on built-in explanation rules. As we have considered the domain of reaction, convec-

tion and diffusion, these explanation rules are driven purely by these individual features. For example, in a reaction-convection-diffusion system, if the sparse model shows importance of features corresponding to $\frac{\partial u}{\partial x}$ and $\frac{\partial^2 u}{\partial x^2}$, tags corresponding to convection and diffusion are activated. Presence of polynomial features are automatically tagged as reaction systems. These tags are then used to explain the underlying mechanisms in the form of causal relationships between the independent variable transformations and dependent variable. Further, for natural language explanation of the feature transformations, a sample template of the following form is created:

The rate variable {} is affected by {}, {}. The system is governed by {}, {}.

A causal model is established relating each rate variable to the corresponding causal variable transformations. Features corresponding to the significant parameters in the estimated process model are first established to estimate the causal relationships. The variables that are transformed to give rise to these significant features are then used to estimate the final causal model. Figure 5.2 shows a causal model estimated for the Lotka-Volterra/Predator-prey model. The figure shows that the rate variable corresponding to u is influenced by u positively and uv negatively. uv in turn is estimated combining u and v . Similarly, the rate variable corresponding to v is affected by uv positively and v negatively. This requires the symbolic mapping of the estimated features and parameters, to the actual output rate variable. These kinds of directed causal graphs, in conjunction with graph-theoretic algorithms [4, 5, 113], can be used for more complex reaction networks, and chemical processing for applications such as fault detection and detection.

5.2 Results and Discussion

In this study, we have considered prototypical model systems (temporal and spatio-temporal) commonly featured in many physics and chemical engineering textbooks that

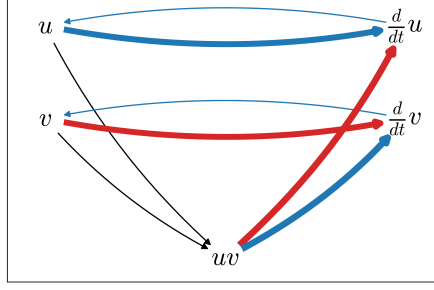


Figure 5.2: Causal graph for a predator prey model with data provided for t , u (Prey fraction), v (Predator fraction). Red corresponds to negative parameters, and blue corresponds to positive parameters. As is seen, $\frac{du}{dt}$ increases with more u , but is negatively impacted due to a second order effect of uv .

teach modeling techniques. The following subsections discuss the performance of XAI-MEG for these model systems.

5.2.1 Case Study 1: Simple Harmonic Motion

Simple harmonic motion is one of the most important prototypes with applications in many different areas. Here the position x and the velocity v , as functions of time, are required to model the system. This yields the coupled system of first-order ordinary differential equations,

$$\begin{aligned}\frac{dx}{dt} &= v \\ \frac{dv}{dt} &= -kx\end{aligned}$$

We simulated the system with $k = 0.05$ and fed the position and velocity data (as a function of time) to XAI-MEG.

The features identified by the system, using its knowledge base of allowed elementary functions based on first-principles knowledge, include transformations involving linear and second order terms in position (x) and velocity (v). From Figure 5.3, it is seen that in the identified model, position contributes the most to the rate of change of velocity, and velocity

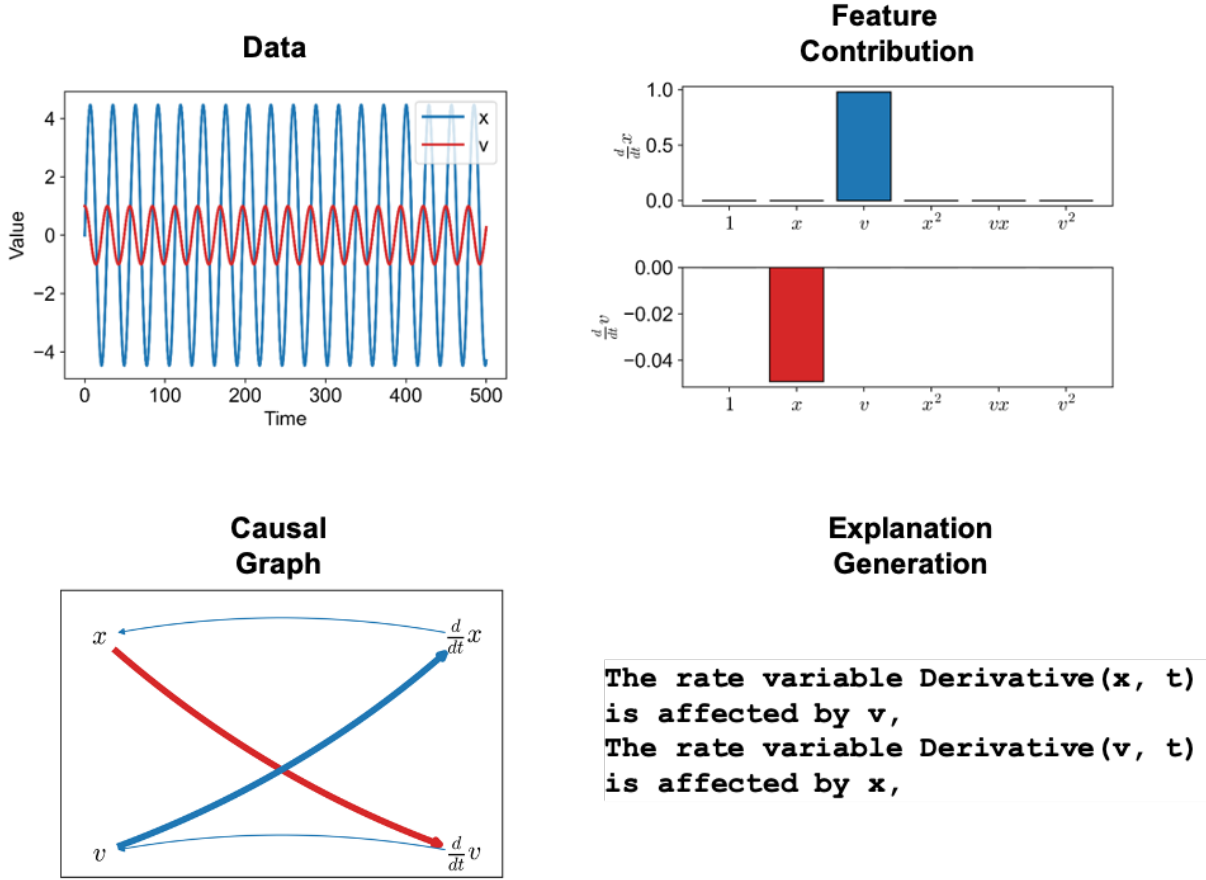


Figure 5.3: Simple Harmonic Motion - Inferences by XAI MEG

directly affects the rate of change of position (as expected from first-principles). XAI-MEG correctly identifies the model, listed above as the two coupled ODEs. The causal graph automatically generated for the system reflects the same. XAI-MEG identifies the *positive* relationship between v and **Derivative(x , t)** (denoted in blue), and the *negative* relationship between x and **Derivative(v , t)** (denoted in red). Finally, a simple explanation is generated stating that **Derivative(x , t)** is affected by v and **Derivative(v , t)** is affected by x .

5.2.2 Case Study 2: Damped Simple Harmonic Motion

Here we add a velocity dependent damping term ($-\zeta v$) to the dynamics of the simple harmonic motion, giving rise to,

$$\begin{aligned}\frac{dx}{dt} &= v \\ \frac{dv}{dt} &= -kx - \zeta v\end{aligned}$$

This kind of model forms is generally seen in dissipative systems, where the oscillatory modes decay over time. We simulated the system with $k = 0.05, \zeta = 0.01$.

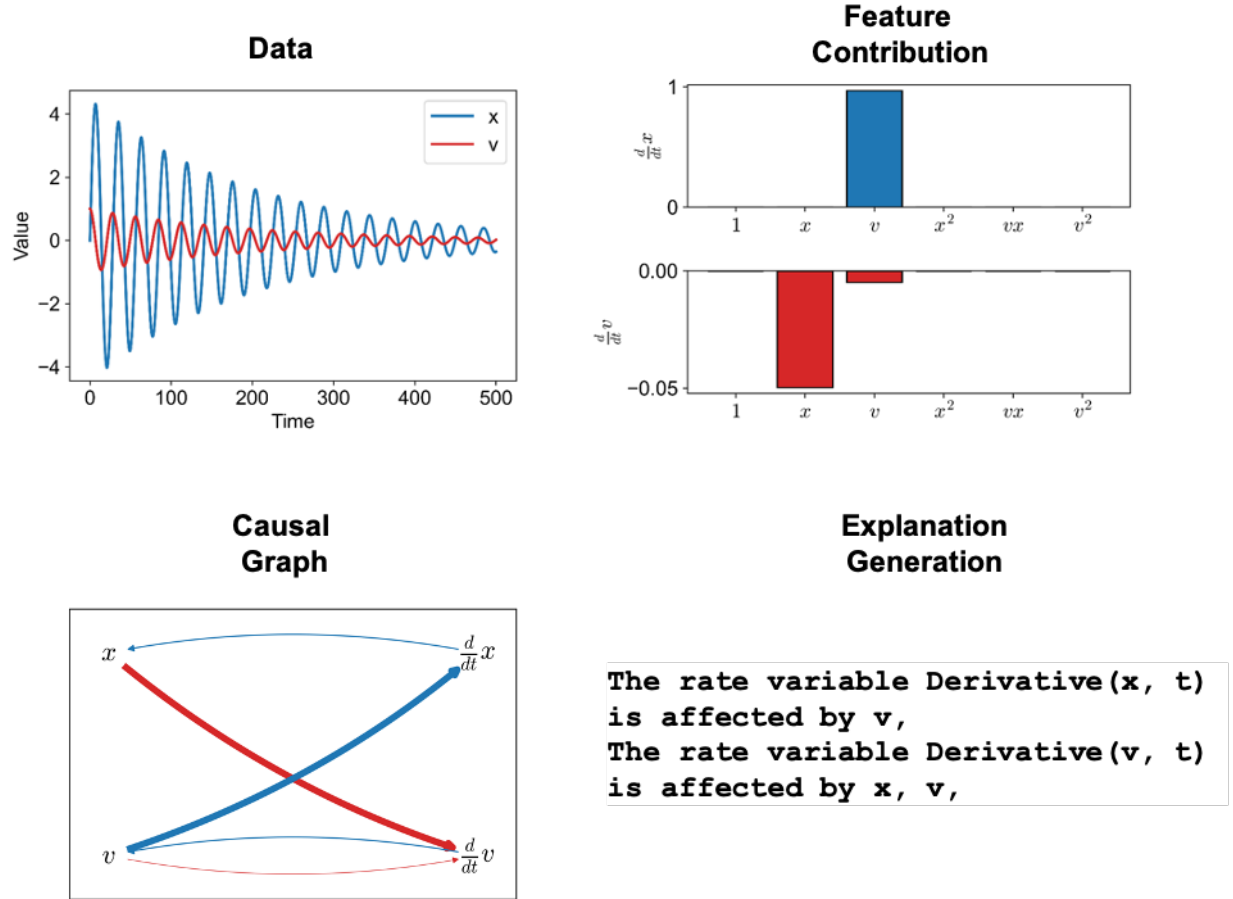


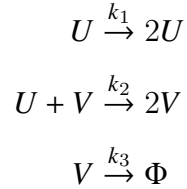
Figure 5.4: Damped Simple Harmonic Motion - Inferences by XAI MEG

The simulated dynamical data corresponding to the time, position, and velocity (similar

to SHM) are provided to XAI-MEG. Again, XAI-MEG correctly identifies the model equations that we used in the simulation. The model identification shows an additional negative contribution of the velocity (v) to the rate of change of velocity ($\frac{dv}{dt}$) (Figure 5.4: Feature Contribution). The weights of the different contributors to the dynamics are shown in the causal graph, especially for the rate of change of velocity. A higher contribution is seen w.r.t. to the position, with low contribution from the velocity feature in the velocity dynamics (Figure 5.4: Causal Graph). Based on these estimated contributions, the explanations are also generated about the causal relationships (Figure 5.4: Explanation Generation).

5.2.3 Case Study 3: Lotka-Volterra System

Lotka-Volterra system [114] shows a competition between two components (similar to a coupled nonlinear reaction setup), where species U and species V react with each other in the following form,



The same system is used to model the dynamics of the population of the predator and prey, where the prey (U) grows in the case of abundance of resources, exponentially, and the presence of predator (V) results in the depletion of the prey population. The predator consumes the prey to grow its population, and in the absence of the prey population deplete over time. This results in a two-component ordinary differential equation of the form,

$$\begin{aligned} \frac{du}{dt} &= k_1u - k_2uv \\ \frac{dv}{dt} &= k_2uv - k_3v \end{aligned}$$

This is one of the prototypical case studies, where we see nonlinear contributions. We simulate the system with $k_1 = k_2 = k_3 = 0.05$.

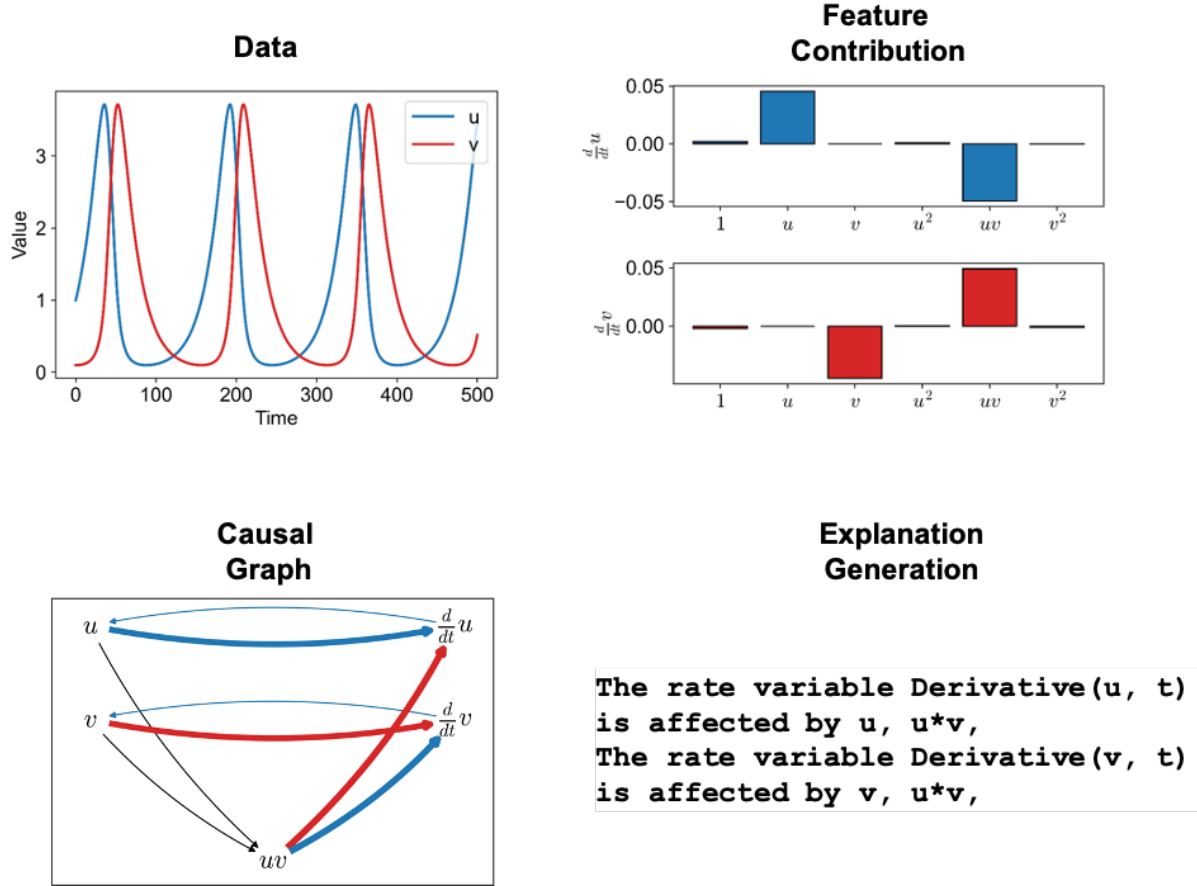
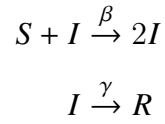


Figure 5.5: Lotka Volterra Model- Inferences by XAI MEG

XAI-MEG automatically examines linear and quadratic features from its knowledge base that correspond to different rate expressions. From Figure 5.5, it is seen that the XAI-MEG correctly identifies the model. It recognizes that u and uv terms contribute most for the dynamics of u and v , and uv term contributes most for the dynamics of v . These significant terms are also seen in the explanations in the form of the causal graph (Figure 5.5: Causal Graph) and the natural language explanations (Figure 5.5: Explanation Generation). As mentioned in the previous section, these terms correspond to the growth, decay, and the competition between the predator and the prey.

5.2.4 Case Study 4: Compartmental Models in Epidemiology – S-I-R model

Similar to the competing dynamics in the Lotka-Volterra system, the S-I-R model for disease spread [115] shows similar competing dynamics between the susceptible population and infected population. This model is used for modeling the dynamics of the spread of a disease considering the dynamics of the susceptible population (S), the infected population (I) and the recovered population (R),



These result in the system of ordinary differential equations,

$$\begin{aligned} \frac{dS}{dt} &= -\beta SI \\ \frac{dI}{dt} &= \beta SI - \gamma I \\ \frac{dR}{dt} &= \gamma I \end{aligned}$$

The system is simulated for $\beta = 0.05, \gamma = 0.01$.

For this case, interactions up to the second order are allowed, and the system correctly identifies the main modes of operation of the system (Figure 5.6: Feature Contribution). It is seen that the rate of change of the susceptible population is correctly identified to be dependent on the second order term involving the susceptible population (S) and the infected population (I). In the dynamics of I , however, we see additional effects from second order I^2 and IR effects from the model. While these added modes do not contribute as severely to the dynamics as the main mode, SI , it raises a question on whether the dynamics from the compartmental model can be explained by additional interactions within and between population I and the recovered population R . The recovered population dynamics however, is correctly identified to be dependent on the infected population.

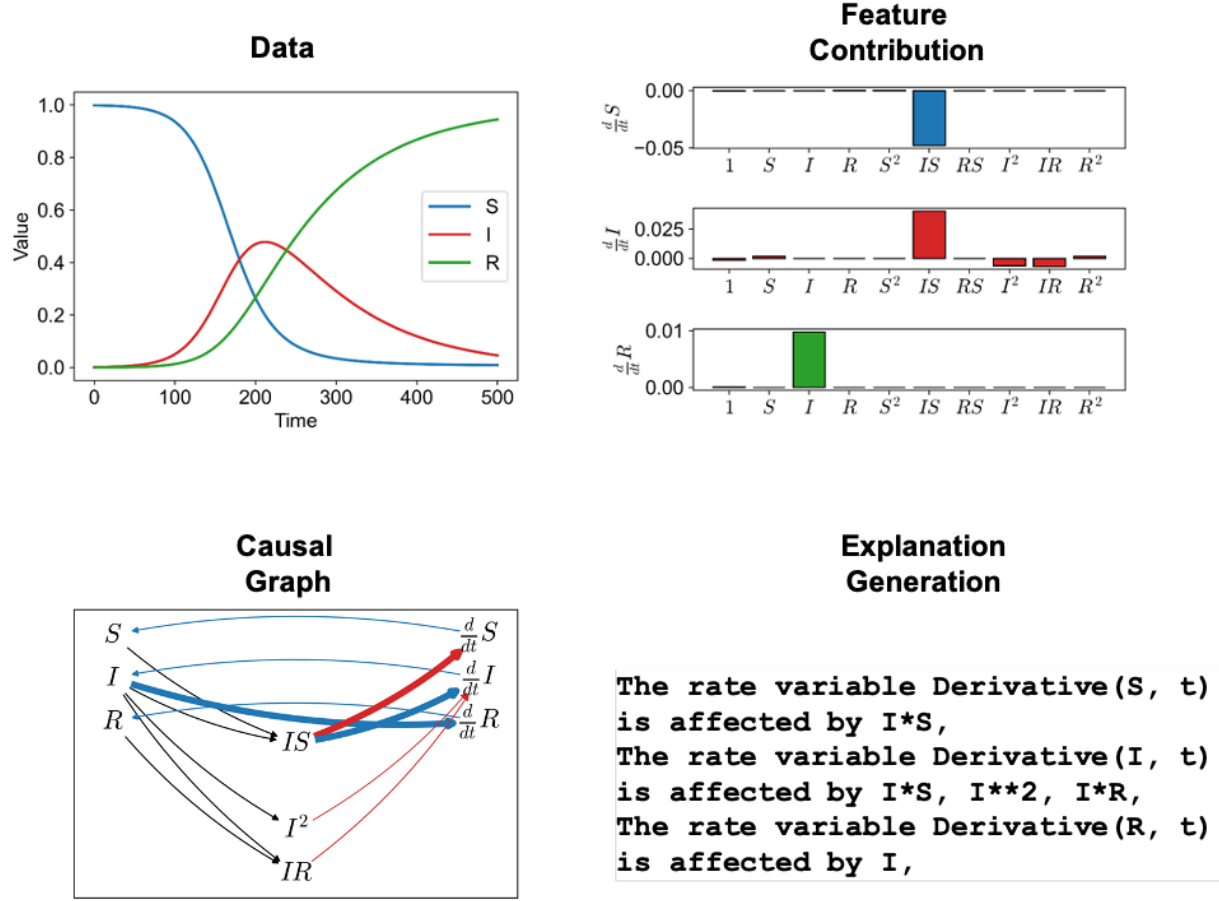


Figure 5.6: SIR Model - Inferences by XAI MEG

5.2.5 Case Study 5: Convection Systems

Convection systems are modeled using a first-order partial differential equation of the form,

$$\frac{\partial u}{\partial t} = -v_x \frac{\partial u}{\partial x}$$

These systems are common in many convection-driven applications of heat and mass. The species of interest, u , in the above case convects along the x axis with a velocity, $v_x = 0.001$ at all points in space.

The spatio-temporal data (Figure 5.7: Data) is provided for variable u of the system.

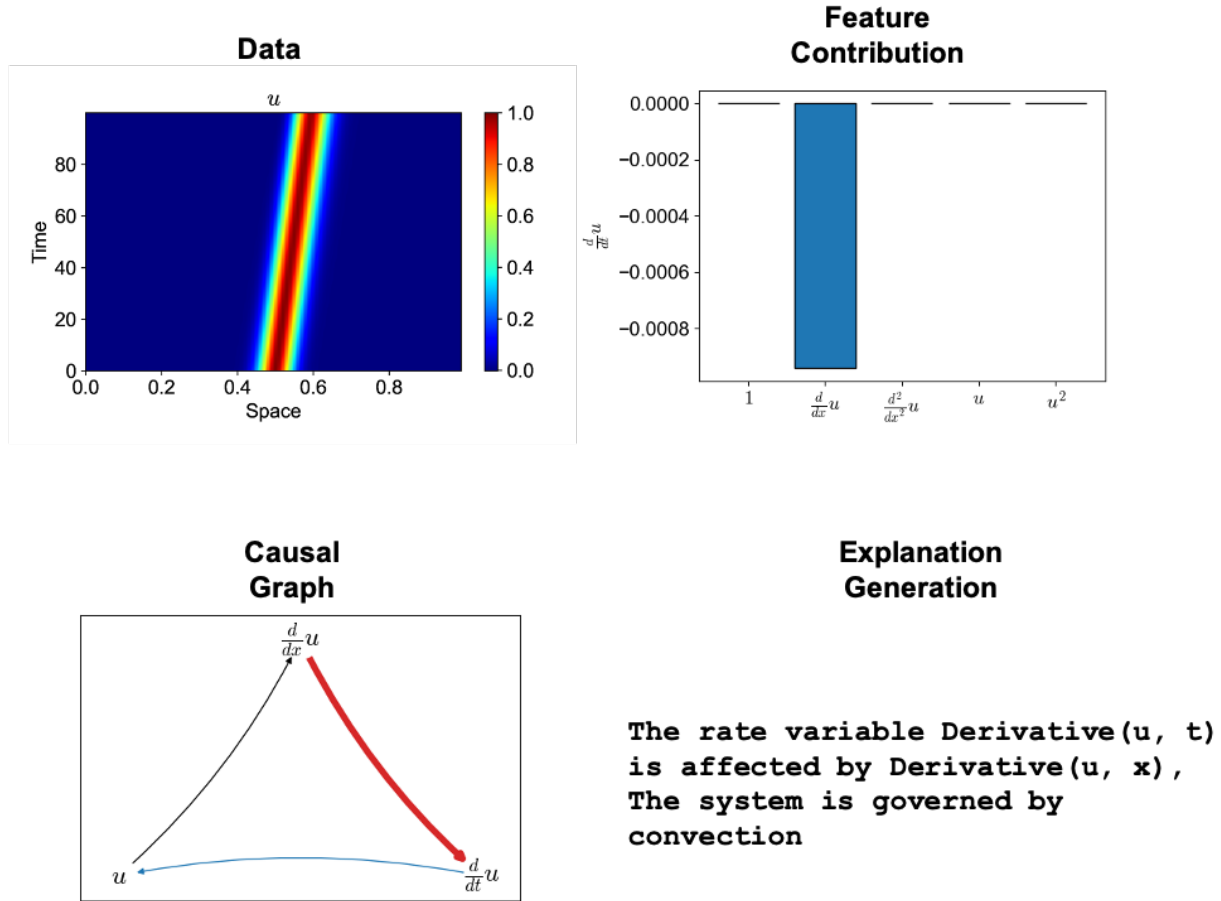


Figure 5.7: Convection Model - Inferences by XAI MEG

Based on this data first-order spatial derivative, second-order spatial derivative, and polynomial features are generated up to order two. The XAI-MEG identifies the dynamics to be only dependent on the first-order derivative feature (Figure 5.7: Feature Contribution). The causal graph shows the same interactions (Figure 5.7: Causal Graph), and the explanation generator (Figure 5.7: Explanation Generation) correctly classifies the system to be a convective system, because of the rate's dependence on purely the first spatial derivative.

5.2.6 Case Study 6: Diffusion Systems

Diffusion systems are systems where components diffuse in space. Diffusion of temperature, species, etc., are governed by a parabolic partial differential equation of the form,

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2}$$

In this study we consider the species diffusing with the diffusion coefficient, $D = 10^{-5}$.

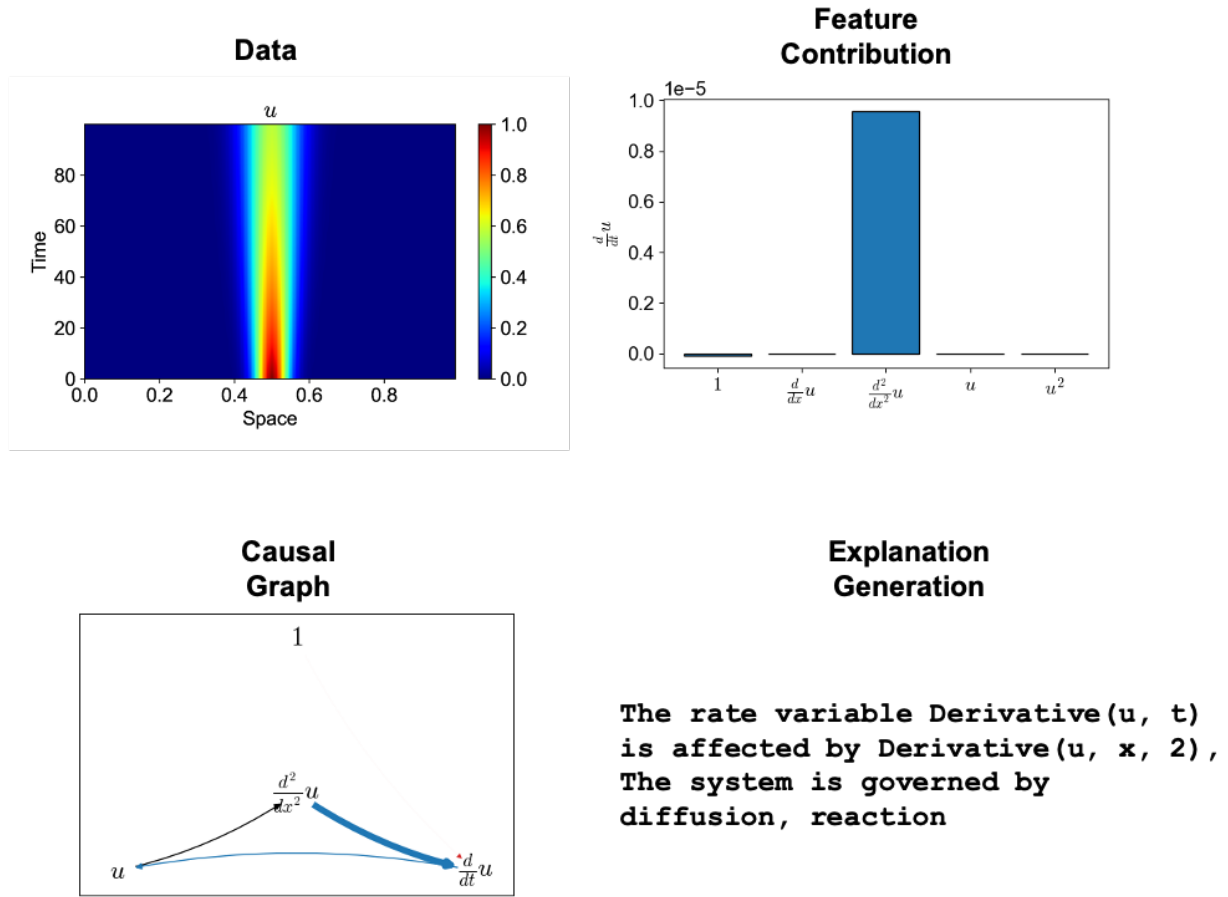


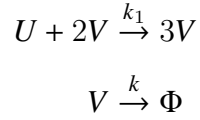
Figure 5.8: Diffusion Model - Inferences by XAI MEG

Similar to the convection data, the spatio-temporal data (Figure 5.8: Data) is sent to the XAI-MEG. The rate is seen to be highly affected by the second-spatial derivative, with minimal dependence on a constant factor which is seen as a reaction class (Figure 5.8: Causal

graph). Based on these identified parameters, the explanation generator shows that the rate variable is dependent on the second-spatial derivative, and hence classifies the system as a reaction diffusion system, while the reaction is in minimal amount.

5.2.7 Case Study 7: Reaction Diffusion Systems

Higher order reaction-diffusion systems are commonly studied in a variety of pattern formation problems, addressed first by Alan Turing [116], and has been used to model a variety of systems in ecology, geology, among other fields of interest. In these systems species diffuse and react in space. We consider a sample case of the Gray-Scott model [117], which is governed by the chemical reactions,



Under the condition that $k_1 = 1$, and U is constantly fed into the reactor at a feed rate of f , and species V is constantly killed at a rate of k to give rise to the inert product Φ , the system results in a set of nonlinear partial differential equations of the form,

$$\begin{aligned} \frac{\partial u}{\partial t} &= D_u \frac{\partial^2 u}{\partial x^2} - uv^2 + f(1 - u) \\ \frac{\partial v}{\partial t} &= D_v \frac{\partial^2 v}{\partial x^2} + uv^2 - (k + f)v \end{aligned}$$

where we consider the system with $D_u = 10^{-1}, D_v = 5 \times 10^{-2}, k = 0.062, f = 0.055$.

The two species data on the concentration of the different species is provided in the form of tensorial data of time and space (Figure 5.9: Data). For the dynamics of u , the first and second spatial derivative features only include derivatives of u , while for v we see only derivatives of v . Further, we see additional polynomial features of u and v and various interactions up to the third order. These third order interaction variables are seen to

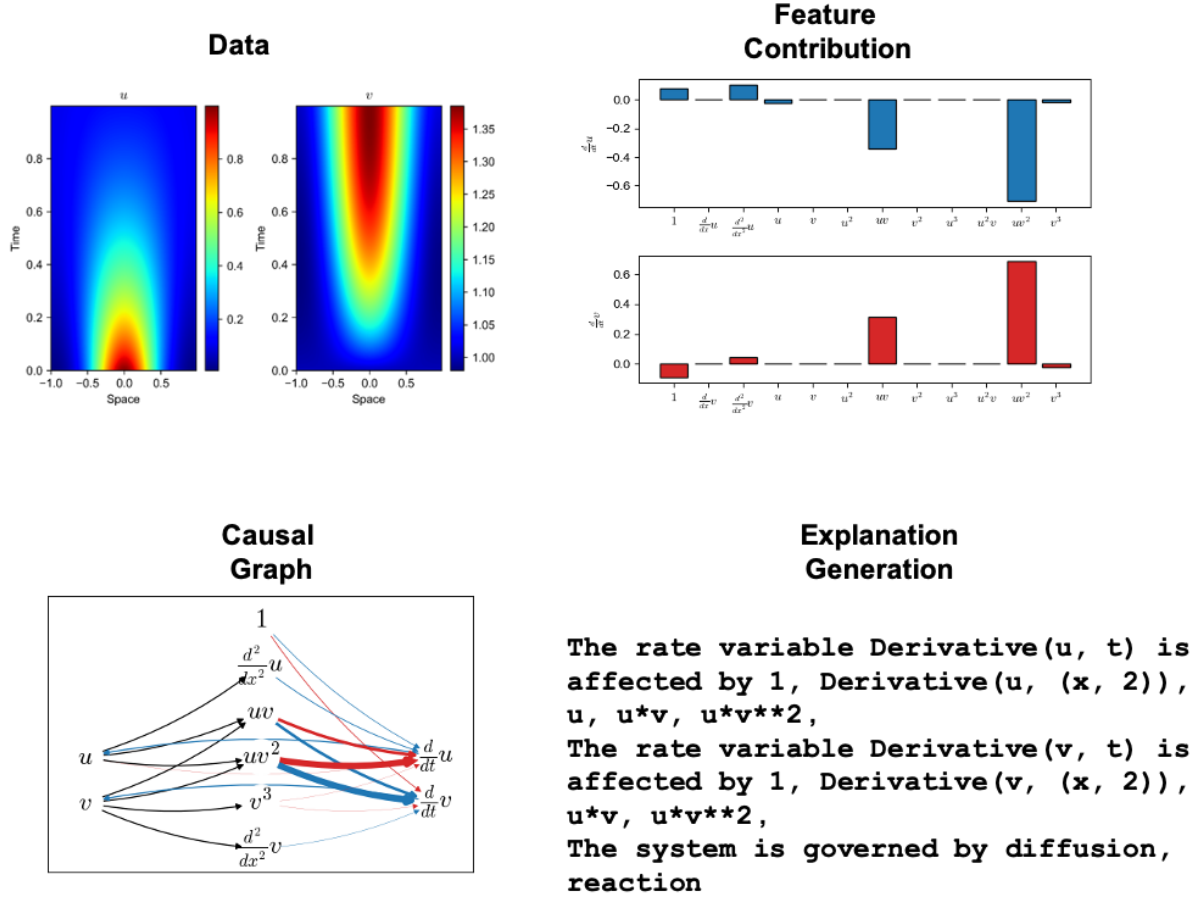


Figure 5.9: Reaction Diffusion Model - Inferences by XAI MEG

affect the rate variables the most. We do see additional second order interactions, of lower magnitude appearing in the final estimated model form (Figure 5.9: Feature contribution). However, the system is correctly tagged based on the features and the parameters, as a reaction, diffusion system.

In summary, the identified relationships are linear in parameters in all these prototypical systems, and hence one can work with simple linear models over the time-frame of experimentation. For simple harmonic motion (SHM), the system correctly identified the feature contributions and the causal map corresponding to the system. Damped SHM also shows similar characteristics with a velocity contribution to the v dynamics, that is involved in the damping. The Lotka-Volterra system is also correctly identified with contributions from the first order terms – u and v – and the second order term of uv . For the SIR model system,

XAI-MEG does not identify the correct contributions for the different terms, but still shows maximum contribution corresponding to the true modes of operation. This is further made understandable through the causal map based on the contribution of the different features (variable transformations) and corresponding parameters.

In the case of the spatio-temporal systems, the data has two independent variables, time and space, where the variables have a particular value for each point in time and space. We see that the model for the convection system was correctly identified to be dependent on the first-derivative features. For diffusion, the feature contributions are similar to the contributions in the true process model. However, there is minimal contribution of concentration-independent reaction mechanism (tagged as '1'). For reaction-diffusion systems, it is seen that the feature contribution of the constant term corresponding to addition of the species is seen in the dynamics of u , with no effect shown due to convection features. A contribution of diffusion is also seen in this case, suggesting that the system is a reaction-diffusion system. Apart from the third order reaction term, an additional effect of a second order reaction is also seen in the reaction-diffusion system. In all these cases, the explanation generation module correctly identifies the underlying mechanisms contributing to the dynamics.

5.3 Major Results

We have currently looked at a variety of case studies with phenomenology of reaction, convection and diffusion. For these prototypical examples, we see that all the underlying phenomenology were identified based on the data from the system, and possible variable transformations. While the underlying model structure is linear in parameters, it is to be noted that such models can still be utilized to generate surrogate models [118, 119]. The advantage however, is that these models are rooted in physics and chemistry, and the surrogate models generated are based on mechanistic variable interactions and not just data driven models like kriging or artificial neural networks. These insights aid in generating causal explanations of the variables and parameters involved during the occurrence of a

fault in the system for each time/space window considered for mechanism identification.

Chapter 6: Discussion

We have looked at the use of machine learning for causal modeling for process systems, generating information about the hidden representations in neural networks, and generating process models for mechanistic explanation generation.

Cause-and-effect reasoning is central to fault diagnosis and hazards analysis of process systems, which require the development and use of causal models. This is also critical to generating explanations of system behavior, particularly under abnormal conditions. Since complex process systems operate at different length and time scales, it is important to model the causality at different levels of granularity. We have proposed a hierarchical approach for causal inferencing that captures causal maps at two levels – the overall plant-level and the subsystem-level. This method decouples the cyclic and non-cyclic effects present in the system and improves the accuracy and reliability of data-based causal inferencing algorithms by reducing the number of spurious predictions. Prior information from the process flowsheet is used to choose variables for each causal map. A matrix transformation algorithm that subsides the effects of indirect causal interactions using the reachability matrix and adjacency matrix ideas is also presented in this work. The proposed approach of hierarchical analysis and matrix transformation can be used with any causal metric. In this article, transfer entropy is used to quantify the causal interactions present between various variables for the purpose of demonstration. We have demonstrated the proposed algorithm on the Tennessee Eastman case study. The causal interactions are represented using a digraph. As cyclic and non-cyclic effects are decoupled to generate a directed acyclic graph for the plant, the fault diagnosis, root cause analysis, and explanation generation would become much easier. Onset of a fault would be accompanied by changes in transfer entropies, which would lead to modulations in both the tiers of causal maps identified. Sequential tracking of these

modulations in the plant-level and subsystem-level causal maps can be used as a metric for identification of faults.

Deep neural networks have evolved into a versatile and powerful tool applicable for a wide range of problems. However, a clear understanding of their internal mechanism that allows for interpretations of internal representations and the parameters involved has not been developed completely yet. We presented our findings from a systematic approach to understanding the operation of neural networks for classification tasks. We began with an individual node and identified a *characteristic equation* that dictates the nature of the node-specific selective activation of the input space. We then studied how several of these activations are processed by nodes in the successive layers to generate complex patterns whose characteristic equations represent *non-linear* curved geometric objects. We showed how wider networks result in several simple characteristic equations and hence simple patterns identified on the input space, while deeper networks result in complex patterns in the input space. Although both types of networks are capable of achieving comparable levels of performance, each has its own advantages and disadvantages. Wider networks result in simple patterns which are fairly generic and can potentially be used for multiple tasks. However, wider networks result in a large number of parameters as compared to deeper networks with comparable performance. On the other hand, deeper networks result in complex shapes that cannot be used as generic templates of patterns.

We also studied the transformation of input space by each layer and examined the degeneracy of parameters in the final layer. Specifically, we identified that there exist an infinite number of configuration of weights of the classifying layer that will result in the same classification accuracy. We also identified that while the weights of hidden layers also exhibit degeneracy, implying that the different weights result in the same characteristic curve on the input space, they result in different activations of the space relative to the characteristic curve and hence different internal representations. All the above entities constitute the individual components of a neural network, which when working together towards minimising

the loss function, result in relevant features being extracted from the input space. These are then used for classification of the input data.

We also identified the source of techniques such as transfer learning, weight normalisation and early stopping that are employed to improve the convergence of training algorithms. We also identified the origin of adversarial examples in neural networks, which result in incorrect interpretations, and in safety-critical applications they can lead to concerns. These behaviors and properties were illustrated with three standard classification examples - the intertwined moons, concentric circles and spirals. Their implications for a fault diagnosis task were demonstrated with a continuously stirred tank reactor. In all the cases, it was observed that building deeper networks, while possibly resulting in a decrease of the loss function, does not necessarily imply a better learning or representation of the data. We also observed that networks with more hidden layers are more likely to identify arbitrarily shaped patterns that need not be representative of the true structure of the data, leading to adversarial challenges.

Most machine learning applications in chemical engineering lack the ability to provide mechanistic insights and causal explanations for the recommendations made by them. Addressing this crucial deficiency requires the integration of symbolic AI with numeric AI, namely, machine learning. We have studied a particular methodology to combine symbolic AI and numeric AI to generate mechanistic insights and causal explanations of the dynamical model of the system.

While the framework is general, we only consider prototypical systems commonly found in many modeling textbooks. One key idea in our approach is that we only allow functional transformations that can be directly linked to fundamental physicochemical principles and mechanisms. This is a first step in formulating a reasoning engine that integrates symbolic manipulation and reasoning with the power of machine learning models.

Our knowledge base, at present, only explores reaction, convection, and diffusion systems, and currently requires full observability. It is true that the requirement for full observability of a system is a challenge. In cases where the system is not fully observable, our approach can

be used to discover reduced-order models. With regards to sparse and noisy data, various denoising techniques and interpolation methods can be used successfully as discussed in [19]. Further improvements would include the estimation for time and spatially varying systems, where parameters could change as a function of time and space. Additional advances need to be made to include algebraic models for reasoning. These directions would also require encoding knowledge and inference based on an ontology of physicochemical model forms and phenomenology.

Epilogue

Machine learning models cannot be blindly used in process systems without having the notion of the underlying causality in these systems. A good rule of thumb is to generate process explanations wherever possible, and using machine learning only in situations where interpolation may be valid. Such data availability is often not possible in a variety of process applications. We work in a field where extrapolation is key. Further research is required to make sure such extrapolation is possible.

References

- [1] V. Venkatasubramanian, “The promise of artificial intelligence in chemical engineering: Is it here, finally?” AICHE Journal, vol. 65, no. 2, pp. 466–478, 2019.
- [2] H. Vedam and V. Venkatasubramanian, “PCA-SDG based process monitoring and fault diagnosis,” Control engineering practice, vol. 7, no. 7, pp. 903–917, 1999.
- [3] R. Suresh, A. Sivaram, and V. Venkatasubramanian, “A hierarchical approach for causal modeling of process systems,” Computers & Chemical Engineering, vol. 123, pp. 170–183, 2019.
- [4] M. R. Maurya, R. Rengaswamy, and V. Venkatasubramanian, “A systematic framework for the development and analysis of signed digraphs for chemical processes. 1. Algorithms and analysis,” Industrial & engineering chemistry research, vol. 42, no. 20, pp. 4789–4810, 2003.
- [5] —, “A systematic framework for the development and analysis of signed digraphs for chemical processes. 2. Control loops and flowsheet analysis,” Industrial & Engineering Chemistry Research, vol. 42, no. 20, pp. 4811–4827, 2003.
- [6] J. Peters, S. Bauer, and N. Pfister, “Causal models for dynamical systems,” arXiv preprint arXiv:2001.04385, 2020.
- [7] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, “Physics-informed machine learning,” Nature Reviews Physics, pp. 1–19, 2021.
- [8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” Journal of Computational Physics, vol. 378, pp. 686–707, 2019.
- [9] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman, “Universal differential equations for scientific machine learning,” arXiv preprint arXiv:2001.04385, 2020.
- [10] R. Roscher, B. Bohn, M. F. Duarte, and J. Garcke, “Explainable machine learning for scientific insights and discoveries,” IEEE Access, vol. 8, pp. 42 200–42 216, 2020.
- [11] M. S. Thiese, Z. C. Arnold, and S. D. Walker, “The misuse and abuse of statistics in biomedical research,” Biochemia medica: Biochemia medica, vol. 25, no. 1, pp. 5–11, 2015.

- [12] M. Jordan, “Artificial intelligence—the revolution hasn’t happened yet,” Medium, Apr, vol. 19, 2018.
- [13] M. Hutson, “Ai researchers allege that machine learning is alchemy,” Science, vol. 360, no. 6388, p. 861, 2018.
- [14] A. Sivaram, L. Das, and V. Venkatasubramanian, “Hidden representations in deep neural networks: Part 1. classification problems,” Computers & Chemical Engineering, vol. 134, p. 106 669, 2020.
- [15] L. Das, A. Sivaram, and V. Venkatasubramanian, “Hidden representations in deep neural networks: Part 2. Regression problems,” Computers & Chemical Engineering, vol. 139, p. 106 895, 2020.
- [16] S. L. Brunton, J. L. Proctor, and J. N. Kutz, “Discovering governing equations from data by sparse identification of nonlinear dynamical systems,” Proceedings of the national academy of sciences, vol. 113, no. 15, pp. 3932–3937, 2016.
- [17] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, “Data-driven discovery of partial differential equations,” Science Advances, vol. 3, no. 4, e1602614, 2017.
- [18] A. Chakraborty, A. Sivaram, L. Samavedham, and V. Venkatasubramanian, “Mechanism discovery and model identification using genetic feature extraction and statistical testing,” Computers & Chemical Engineering, vol. 140, p. 106 900, 2020.
- [19] A. Chakraborty, A. Sivaram, and V. Venkatasubramanian, “AI-DARWIN: A first principles-based model discovery engine using machine learning,” Computers & Chemical Engineering, p. 107 470, 2021.
- [20] P. Judea and D. Mackenzie, The Book of Why: The New Science of Cause and Effect. Basic Books, 2018, ISBN: 978-0465097616.
- [21] C Zhao, M Bhushan, and V Venkatasubramanian, “PHASuite: An automated HAZOP analysis tool for chemical processes: Part I: Knowledge engineering framework,” Process Safety and Environmental Protection, vol. 83, no. 6, pp. 509–532, 2005.
- [22] —, “PHASuite: An automated HAZOP analysis tool for chemical processes: Part II: Implementation and case study,” Process Safety and Environmental Protection, vol. 83, no. 6, pp. 533–548, 2005.
- [23] V. Venkatasubramanian, R. Rengaswamy, and S. N. Kavuri, “A review of process fault detection and diagnosis Part II: Qualitative models and search strategies,” Computers and Chemical Engineering, vol. 27, no. 3, pp. 313–326, 2003.

- [24] P. Suppes, A probabilistic theory of causality. North-Holland Publishing Company Amsterdam, 1970.
- [25] C. W. Granger, “Investigating causal relations by econometric models and cross-spectral methods,” Econometrica: Journal of the Econometric Society, pp. 424–438, 1969.
- [26] —, “Testing for causality: A personal viewpoint,” Journal of Economic Dynamics and control, vol. 2, pp. 329–352, 1980.
- [27] T. Schreiber, “Measuring information transfer,” Physical review letters, vol. 85, no. 2, p. 461, 2000.
- [28] J. Pearl, Causality. Cambridge university press, 2009.
- [29] S. L. Bressler and A. K. Seth, “Wiener–Granger causality: A well established methodology,” Neuroimage, vol. 58, no. 2, pp. 323–329, 2011.
- [30] M. Bauer, J. W. Cox, M. H. Caveness, J. J. Downs, and N. F. Thornhill, “Finding the direction of disturbance propagation in a chemical process using transfer entropy,” IEEE transactions on control systems technology, vol. 15, no. 1, pp. 12–21, 2007.
- [31] P. Duan, F. Yang, T. Chen, and S. L. Shah, “Direct causality detection via the transfer entropy approach,” IEEE transactions on control systems technology, vol. 21, no. 6, pp. 2052–2066, 2013.
- [32] W. Yu and F. Yang, “Detection of causality between process variables based on industrial alarm data using transfer entropy,” Entropy, vol. 17, no. 8, pp. 5868–5887, 2015.
- [33] R. Vicente, M. Wibral, M. Lindner, and G. Pipa, “Transfer entropy – A model-free measure of effective connectivity for the neurosciences,” Journal of computational neuroscience, vol. 30, no. 1, pp. 45–67, 2011.
- [34] L. Barnett, A. B. Barrett, and A. K. Seth, “Granger causality and transfer entropy are equivalent for Gaussian variables,” Physical review letters, vol. 103, no. 23, p. 238 701, 2009.
- [35] M. R. Maurya, R. Rengaswamy, and V. Venkatasubramanian, “A systematic framework for the development and analysis of signed digraphs for chemical processes. 1. Algorithms and analysis,” Industrial & engineering chemistry research, vol. 42, no. 20, pp. 4789–4810, 2003.

- [36] —, “Application of signed digraphs-based analysis for fault diagnosis of chemical process flowsheets,” Engineering Applications of Artificial Intelligence, vol. 17, no. 5, pp. 501–518, 2004.
- [37] J. Thambirajah, L. Benabbas, M. Bauer, and N. F. Thornhill, “Cause-and-effect analysis in chemical processes utilizing XML, plant connectivity and quantitative process history,” Computers & Chemical Engineering, vol. 33, no. 2, pp. 503–512, 2009.
- [38] H. Kantz and T. Schreiber, Nonlinear time series analysis. Cambridge university press, 2004, vol. 7.
- [39] T. Schreiber and A. Schmitz, “Surrogate time series,” Physica D: Nonlinear Phenomena, vol. 142, no. 3-4, pp. 346–382, 2000.
- [40] A. Bathelt, N. L. Ricker, and M. Jelali, “Revision of the Tennessee Eastman process model,” IFAC-PapersOnLine, vol. 48, no. 8, pp. 309–314, 2015.
- [41] J. J. Downs and E. F. Vogel, “A plant-wide industrial process control problem,” Computers & chemical engineering, vol. 17, no. 3, pp. 245–255, 1993.
- [42] T. Larsson, K. Hestetun, E. Hovland, and S. Skogestad, “Self-optimizing control of a large-scale plant: The Tennessee Eastman process,” Industrial & engineering chemistry research, vol. 40, no. 22, pp. 4889–4901, 2001.
- [43] N. L. Ricker, “Decentralized control of the Tennessee Eastman challenge process,” Journal of Process Control, vol. 6, no. 4, pp. 205–221, 1996.
- [44] J. Shi, J. Zhao, X. Liu, L. Chen, and T. Li, “Quantifying direct dependencies in biological networks by multiscale association analysis,” IEEE/ACM Transactions on Computational Biology, 2018.
- [45] R. Kannan and A. K. Tangirala, “Correntropy-based partial directed coherence for testing multivariate granger causality in nonlinear processes,” Physical Review E, vol. 89, no. 6, p. 062 144, 2014.
- [46] V. A. Vakorin, O. A. Krakovska, and A. R. McIntosh, “Confounding effects of indirect connections on causality estimation,” Journal of neuroscience methods, vol. 184, no. 1, pp. 152–160, 2009.
- [47] H. Jiang, R. Patwardhan, and S. L. Shah, “Root cause diagnosis of plant-wide oscillations using the concept of adjacency matrix,” Journal of Process Control, vol. 19, no. 8, pp. 1347–1354, 2009.
- [48] M. Newman, Networks: An introduction. Oxford university press, 2010.

- [49] N. Young, “The rate of convergence of a matrix power series,” Linear Algebra and its Applications, vol. 35, pp. 261–278, 1981.
- [50] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, “Understanding neural networks through deep visualization,” arXiv preprint arXiv:1506.06579, 2015.
- [51] R. Setiono and H. Liu, “Understanding neural networks via rule extraction,” in IJCAI, vol. 1, 1995, pp. 480–485.
- [52] J. D. Olden and D. A. Jackson, “Illuminating the “black box”: A randomization approach for understanding variable contributions in artificial neural networks,” Ecological modelling, vol. 154, no. 1-2, pp. 135–150, 2002.
- [53] N. Tishby, F. C. Pereira, and W. Bialek, “The information bottleneck method,” arXiv preprint physics/0004057, 2000.
- [54] N. Tishby and N. Zaslavsky, “Deep learning and the information bottleneck principle,” in 2015 IEEE Information Theory Workshop (ITW), IEEE, 2015, pp. 1–5.
- [55] R. Shwartz-Ziv and N. Tishby, “Opening the black box of deep neural networks via information,” arXiv preprint arXiv:1703.00810, 2017.
- [56] S. J. Pan and Q. Yang, “A survey on transfer learning,” IEEE Transactions on knowledge and data engineering, vol. 22, no. 10, pp. 1345–1359, 2009.
- [57] M. E. Taylor and P. Stone, “Transfer learning for reinforcement learning domains: A survey,” Journal of Machine Learning Research, vol. 10, no. Jul, pp. 1633–1685, 2009.
- [58] K. Weiss, T. M. Khoshgoftaar, and D. Wang, “A survey of transfer learning,” Journal of Big data, vol. 3, no. 1, p. 9, 2016.
- [59] M. Talo, U. B. Baloglu, Ö. Yıldırım, and U. R. Acharya, “Application of deep transfer learning for automated brain abnormality classification using mr images,” Cognitive Systems Research, vol. 54, pp. 176–188, 2019.
- [60] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, “Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning,” IEEE transactions on medical imaging, vol. 35, no. 5, pp. 1285–1298, 2016.
- [61] K. Gopalakrishnan, S. K. Khaitan, A. Choudhary, and A. Agrawal, “Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection,” Construction and Building Materials, vol. 157, pp. 322–330, 2017.

- [62] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in Proceedings of the thirteenth international conference on artificial intelligence 2010, pp. 249–256.
- [63] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in Advances in Neural Information Processing Systems 2016, pp. 901–909.
- [64] L. Prechelt, "Early stopping-but when?" In Neural Networks: Tricks of the trade, Springer, 1998, pp. 55–69.
- [65] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in Advances in neural information processing systems, 2001, pp. 402–408.
- [66] V. Venkatasubramanian and K. Chan, "A neural network methodology for process fault diagnosis," AIChE Journal, vol. 35, no. 12, pp. 1993–2002, 1989.
- [67] K. Watanabe, I. Matsuura, M. Abe, M. Kubota, and D. Himmelblau, "Incipient fault diagnosis of chemical processes via artificial neural networks," AIChE journal, vol. 35, no. 11, pp. 1803–1812, 1989.
- [68] J. Hoskins, K. Kaliyur, and D. M. Himmelblau, "Fault diagnosis in complex chemical plants using artificial neural networks," AIChE Journal, vol. 37, no. 1, pp. 137–141, 1991.
- [69] J. Fan, M. Nikolaou, and R. E. White, "An approach to fault diagnosis of chemical processes via neural networks," AIChE Journal, vol. 39, no. 1, pp. 82–88, 1993.
- [70] M. Askarian, G. Escudero, M. Graells, R. Zarghami, F. Jalali-Farahani, and N. Mostoufi, "Fault diagnosis of chemical processes with incomplete observations: A comparative study," Computers & chemical engineering, vol. 84, pp. 104–116, 2016.
- [71] P. Jiang, Z. Hu, J. Liu, S. Yu, and F. Wu, "Fault diagnosis based on chemical sensor data with an active deep neural network," Sensors, vol. 16, no. 10, p. 1695, 2016.
- [72] A. Shokry, M. H. Ardakani, G. Escudero, M. Graells, and A. Espuña, "Dynamic kriging based fault detection and diagnosis approach for nonlinear noisy dynamic processes," Computers & Chemical Engineering, vol. 106, pp. 758–776, 2017.
- [73] H. Gharahbagheri, S. Imtiaz, and F. Khan, "Root cause diagnosis of process fault using kpca and bayesian network," Industrial & Engineering Chemistry Research, vol. 56, no. 8, pp. 2054–2070, 2017.

- [74] Z. Zhang and J. Zhao, “A deep belief network based fault diagnosis model for complex chemical processes,” Computers & Chemical Engineering, vol. 107, pp. 395–407, 2017.
- [75] H. Wu and J. Zhao, “Deep convolutional neural network model based chemical process fault diagnosis,” Computers & Chemical Engineering, vol. 115, pp. 185–197, 2018.
- [76] R Vaidyanathan and V Venkatasubramanian, “On the nature of fault space classification structure developed by neural networks,” Engineering Applications of Artificial Intelligence, vol. 5, no. 4, pp. 289–297, 1992.
- [77] K. E. Pilario, Feedback-controlled cstr process for fault simulation, (<https://www.mathworks.com/matlabcentral/fileexchange/66189-feedback-controlled-cstr-process-for-fault-simulation>), MATLAB Central File Exchange, Retrieved June 10, 2019, 2019.
- [78] N. Dalvi, P. Domingos, S. Sanghai, D. Verma, et al., “Adversarial classification,” in Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data, ACM, 2004, pp. 99–108.
- [79] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” arXiv preprint arXiv:1412.6572, 2014.
- [80] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” arXiv preprint arXiv:1607.02533, 2016.
- [81] S. Gu and L. Rigazio, “Towards deep neural network architectures robust to adversarial examples,” arXiv preprint arXiv:1412.5068, 2014.
- [82] D. M. Elbrächter, J. Berner, and P. Grohs, “How degenerate is the parametrization of neural networks with the relu activation function?” Advances in Neural Information Processing Systems, vol. 32, 2019.
- [83] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
- [84] S. Zagoruyko and N. Komodakis, “Wide residual networks,” arXiv preprint arXiv:1605.07146, 2016.
- [85] A. Sivaram, L. Das, and V. Venkatasubramanian, “Hidden representations in deep neural networks: Part 1. classification problems,” Submitted to Computers and Chemical Engineering, 2019.

- [86] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," IEEE Transactions on neural networks, vol. 1, no. 1, pp. 4–27, 1990.
- [87] S. Chen, S. Billings, and P. Grant, "Non-linear system identification using neural networks," International journal of control, vol. 51, no. 6, pp. 1191–1214, 1990.
- [88] J. Schmidt, M. R. Marques, S. Botti, and M. A. Marques, "Recent advances and applications of machine learning in solid-state materials science," npj Computational Materials, vol. 5, no. 1, pp. 1–36, 2019.
- [89] Y. Pan, J. Jiang, and Z. Wang, "Quantitative structure–property relationship studies for predicting flash points of alkanes using group bond contribution method with back-propagation neural network," Journal of hazardous materials, vol. 147, no. 1-2, pp. 424–430, 2007.
- [90] N. Artrith, T. Morawietz, and J. Behler, "High-dimensional neural-network potentials for multicomponent systems: Applications to zinc oxide," Physical Review B, vol. 83, no. 15, p. 153 101, 2011.
- [91] G. Schmitz, I. H. Godtliessen, and O. Christiansen, "Machine learning for potential energy surfaces: An extensive database and assessment of methods," The Journal of Chemical Physics, vol. 150, no. 24, p. 244 113, 2019.
- [92] D. T. Jones, "Protein secondary structure prediction based on position-specific scoring matrices," Journal of molecular biology, vol. 292, no. 2, pp. 195–202, 1999.
- [93] H. Gao, T. J. Struble, C. W. Coley, Y. Wang, W. H. Green, and K. F. Jensen, "Using machine learning to predict suitable conditions for organic reactions," ACS central science, vol. 4, no. 11, pp. 1465–1476, 2018.
- [94] A. Parlak, Y. Islamoglu, H. Yasar, and A. Egrisogut, "Application of artificial neural network to predict specific fuel consumption and exhaust temperature for a diesel engine," Applied Thermal Engineering, vol. 26, no. 8-9, pp. 824–828, 2006.
- [95] A. Sundaram, P. Ghosh, J. M. Caruthers, and V. Venkatasubramanian, "Design of fuel additives using neural networks and evolutionary algorithms," AIChE Journal, vol. 47, no. 6, pp. 1387–1406, 2001.
- [96] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in European conference on computer vision, Springer, 2014, pp. 818–833.
- [97] S. Ruder, "An overview of gradient descent optimization algorithms," arXiv preprint arXiv:1609.04747, 2016.

- [98] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” Mathematics of control, sig, vol. 2, no. 4, pp. 303–314, 1989.
- [99] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” Neural networks, vol. 2, no. 5, pp. 359–366, 1989.
- [100] K. Hornik, “Some new results on neural network approximation,” Neural networks, vol. 6, no. 8, pp. 1069–1072, 1993.
- [101] T. Chen and H. Chen, “Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems,” IEEE Transactions on Neural Networks, vol. 6, no. 4, pp. 911–917, 1995.
- [102] B. Hanin, “Universal function approximation by deep neural nets with bounded width and relu activations,” arXiv preprint arXiv:1708.02691, 2017.
- [103] D. Yarotsky, “Error bounds for approximations with deep relu networks,” Neural Networks, vol. 94, pp. 103–114, 2017.
- [104] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” Journal of Machine Learning Research, vol. 12, no. Jul, pp. 2121–2159, 2011.
- [105] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6, 2014.
- [106] G. Hinton, “Neural networks for machine learning - lecture 6a - overview of mini-batch gradient descent.,” 2012.
- [107] B. R. Bakshi and G. Stephanopoulos, “Wave-net: A multiresolution, hierarchical neural network with localized learning,” AIChE Journal, vol. 39, no. 1, pp. 57–81, 1993.
- [108] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” arXiv preprint arXiv:1207.0580, 2012.
- [109] M. Carlsson, “On convex envelopes and regularization of non-convex functionals without moving global minima,” Journal of Optimization Theory and Applications, vol. 183, no. 1, pp. 66–84, 2019.
- [110] M. Schmidt and H. Lipson, Eureqa, version 0.98 beta, 2013.
- [111] B. Babu and S. Karthik, “Genetic programming for symbolic regression of chemical process systems,” Engineering Letters, vol. 14, no. 2, pp. 42–55, 2007.

- [112] F. Van Breugel, J. N. Kutz, and B. W. Brunton, “Numerical differentiation of noisy data: A unifying multi-objective optimization framework,” IEEE Access, vol. 8, pp. 196 865–196 877, 2020.
- [113] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, “A review of process fault detection and diagnosis: Part i: Quantitative model-based methods,” Computers & chemical engineering, vol. 27, no. 3, pp. 293–311, 2003.
- [114] A. J. Lotka, “Contribution to the theory of periodic reactions,” The Journal of Physical Chemistry, vol. 14, no. 3, pp. 271–274, 2002.
- [115] D. Smith, L. Moore, et al., “The SIR model for spread of disease-the differential equation model,” Convergence, 2004.
- [116] A. M. Turing, “The chemical basis of morphogenesis,” Bulletin of mathematical biology, vol. 52, no. 1, pp. 153–197, 1990.
- [117] J. E. Pearson, “Complex patterns in a simple system,” Science, vol. 261, no. 5118, pp. 189–192, 1993.
- [118] K. McBride and K. Sundmacher, “Overview of surrogate modeling in chemical process engineering,” Chemie Ingenieur Technik, vol. 91, no. 3, pp. 228–239, 2019.
- [119] M. A. Bouhlel, J. T. Hwang, N. Bartoli, R. Lafage, J. Morlier, and J. R. Martins, “A python surrogate modeling framework with derivatives,” Advances in Engineering Software, vol. 135, p. 102 662, 2019.