Quadratic Convolutional and Physics-Informed Neural Networks with Applications to System Theory

Zachary Yetman Van Egmond

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science in Electrical and Computer Engineering

Concordia University

Montréal, Québec, Canada

August 2025

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: Zachary Yetman Van Egmond

Entitled: Quadratic convolutional and physics-informed neural networks with applications to system theory

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Electrical and Computer Engineering

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

_____ Chair

Dr. R. Selmic

_____ External Examiner

Dr. A. Mohammadi

_____ Examiner

Dr. R. Selmic

_____ Supervisor

Dr. L. Rodrigues

Approved by _____

Dr. A. Hamou-Lhadj, Chair

Department of Electrical and Computer Engineering

_____ 2025 _____

Dr. M. Debbabi, Dean

Faculty of Engineering and Computer Science

Abstract

Quadratic convolutional and physics-informed neural networks
with applications to system theory

Zachary Yetman Van Egmond

Motivated by a growing need for explainable artificial intelligence, this thesis explores quadratic neural networks (QNNs) as a solution to many issues that limit the application of neural networks to safety-critical systems. Specifically, this thesis shows that the training of both a two-layer convolutional quadratic neural network (CQNN) and a two-layer quadratic physics-informed neural network (QPINN), with no regularization and constraints added on their activation function parameters and the norm of their weights, can be formulated as a least squares problem. Using this method, an analytic expression for the globally optimal weights is obtained, alongside a quadratic input-output mapping for the network. These properties make the network a viable tool in system theory by enabling formal analysis. Furthermore, compared to backpropagation-trained networks, the least squares training significantly reduces both training time and the number of hyperparameters that the designer must select. Additionally, it is proven that two-layer QNNs become universal approximators when a monomial lifted input of sufficiently high degree is used. Finally, ensemble QNNs (EQNNs) are proposed to train multiple QNNs across the training space to increase the representational capacity of QNNs.

Acknowledgements

Firstly, I want to thank my family for their unwavering love and support which kept me going even in the toughest times. To my mom and dad, I am forever appreciative of the wisdom you gave and the love you provided, you are both an inspiration. To my brother Ed, your creativity and talent are a never-ending source of inspiration, you make me want to better myself to be the best older brother I can be.

To Linh, I am forever grateful for your immense love and kindness. Spending time with you gives me happiness and was what I looked forward to the most after long days of work.

I want to extend my sincerest gratitude to my supervisor, Dr. Luis Rodrigues, who continuously pushed me to be the best version of myself. The work I've achieved over the past two years would not have been possible without your guidance and mentorship.

I also want to thank all the colleagues I've had the pleasure of working alongside during my master's. To Steven, Camilo, Karolina, Lucas, Mohammad, Mahmood, Soroush, Sonali, and Giuseppe, your presence brought so much life to the lab, and our discussions always helped elevate my work.

I also want to give my sincerest thanks to all my friends for being just the absolute best. In times of high stress and intense workloads, it was spending time with you guys that kept me laughing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Neural networks are a class of machine learning models that take their inspiration from the biological brain. Although they trace their origins to the 1940s, the foundations of modern neural network theory were established throughout the 1980s [1]. Neural networks are built from components called neurons. Each neuron takes a series of inputs, multiplies each by a gain (which are collectively called the weights of the network), before summing them and passing the result through an activation function to produce an output. Multiple neurons are stacked to form layers, which in turn are stacked to form the complete network. The conventional depictions of a neuron, a layer, and a network are shown in Figure 1.1. Network training occurs by updating the weights (typically in an iterative fashion) to minimize some loss function applied to a set of training data. As a data-driven tool, neural networks have shown great potential in prediction and classification tasks in many sectors, such as agriculture, energy, and finance [2]. Despite this success, neural networks present several shortcomings that limit their application to certain engineering problems, specifically to applications involving safety-critical systems.

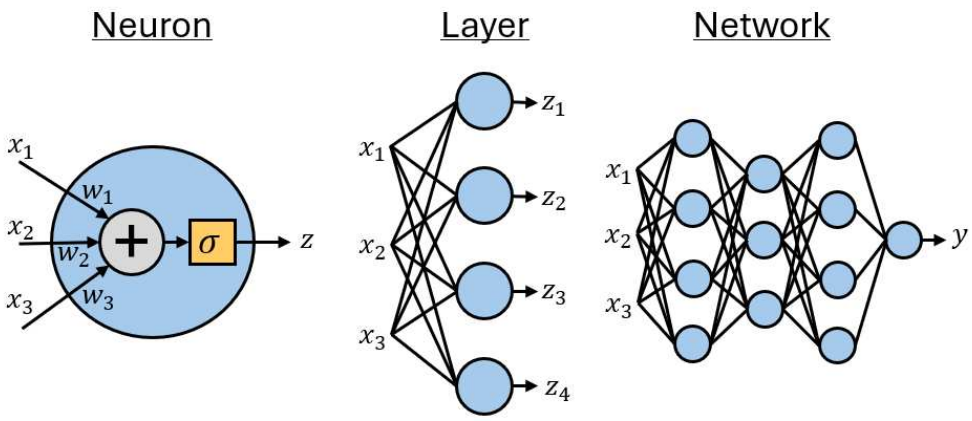


Figure 1.1: Structures in a neural network: neuron, layer, and complete network

Some of the main challenges that neural networks face are:

1. **Explainability:** One of the primary issues limiting the application of neural networks to safety-critical systems is the difficulty in making definitive statements about the behavior and predictions of the networks. This is often referred to as the explainability problem in the broader context of artificial intelligence (AI). Due to their depth and nonlinearity, deep neural networks are oftentimes viewed as black box systems, with the outputs often being highly sensitive to disturbances in the input [3]. Furthermore, despite being deterministic systems, the inability to test the networks over the entire input space requires them to be analyzed as though they were a stochastic system [4]. These issues present challenges to the implementation of neural networks in safety-critical applications, where formal verification is often required to ensure certification.

2. **Global Optimality:** Due to their depth and nonlinearity, the loss function of neural networks is typically non-convex. This results in their optimization algorithms (typically variations of gradient descent) potentially getting caught in local minima, making it difficult to find the globally optimal weights of the network.

3. **Architecture and Hyperparameters:** The selection of a network architecture, that is to say, the number of layers and neurons, and the choice of activation function, are

all based primarily on heuristics. Despite two-layer networks being universal approximators from a theoretical perspective [5, 6], in practice, the proper selection of network architecture plays an important role in the performance [7]. Furthermore, the use of gradient descent for training introduces hyperparameters such as learning rate, momentum, and even the choice of gradient descent algorithm, which must also be selected and tuned.

4. **Training Time:** Finally, due to its iterative nature, backpropagation training is time-consuming and computationally demanding. Not only is this undesirable for the user, but it is also less sustainable from an environmental perspective [8].

Despite these difficulties, there is a clear interest in the adoption of neural network technologies. Using the aviation sector as an example, efforts to establish standards and regulations surrounding artificial intelligence have been shown by government agencies such as the European Union Aviation Safety Agency (EASA), in particular with their release of an aviation artificial intelligence roadmap [9]. This is further seen with the development of AI committees by aviation standards organizations such as SAE and EUROCAE [10, 11]. These efforts show the interest of the engineering community in explainable AI.

A type of network that has been of recent interest that addresses many of these concerns are quadratic neural networks (QNNs). Two-layer QNNs possess the following properties [12, 13]: an analytic input-output expression as a quadratic form, a convex training formulation providing a guarantee of globally optimal weights, an architecture that is a by-product of the training, and reduced training times. All of these properties motivate a deeper study of QNNs with applications to system theory.

1.2 Literature Review

1.2.1 Quadratic Neural Networks

Quadratic neural networks (QNNs) are networks that use a second-order polynomial as their activation function. An important distinction needs to be made between QNNs and networks that use quadratic neural units (QNUs), which are often also referred to as quadratic neural networks in the literature. Unlike the QNNs discussed in this thesis, QNU networks use a quadratic form for their pre-activation function weights, and do not necessarily use a second-order polynomial activation function (see [14, 15, 16] for examples of QNU networks).

Although quadratic activation functions have not seen as much widespread success as others, such as ReLU, Tanh, or Sigmoid, QNNs have been used to explore theoretical properties of neural networks. For example, two-layer QNNs were used to show that over-parametrization of networks causes gradient descent algorithms to both converge to global optima [17, 18] and prioritize simpler models [19]. Furthermore, Livni et al. [20] explored the expressivity of deep networks with polynomial activation functions by showing they could approximate two-layer sigmoidal networks. A key development surrounding QNNs occurred in 2021, when Bartan and Pilanci [12, 13] showed that the training of a two-layer QNN with constraints placed on its weights and regularization is a convex problem with guarantees of finding the globally optimal weights. Furthermore, although it is a known result that quadratic activation functions result in networks with polynomial input-output mappings [20], Bartan and Pilanci showed that for two-layer QNNs the input-output mapping of the network is a quadratic form. Extending this work, Rodrigues [21] showed that without regularization the training of a QNN could be formulated as a least squares problem. This allowed for the analytic training of QNNs, which drastically reduced training times. Further work on QNNs presented by Rodrigues and Givigi [22] demonstrated their potential to be used for system identification and controller design with Lyapunov-based stability guarantees.

The quadratic form of the input-output mapping of two-layer QNNs is also a useful

property from the perspective of system theory. In particular, it has been shown that any nonlinear system can be represented equivalently by quadratic dynamics in a higher-dimensional, yet finite, state space [23, 24, 25]. Furthermore, results are known for evaluating the region of asymptotic stability of quadratic systems [26].

1.2.2 Single-Epoch Training of Convolutional Neural Networks

Non-iterative neural network training, sometimes referred to as single epoch training, are methods that allow for the network weights to be determined in an analytic fashion. Typical backpropagation training is a time-consuming and computationally intensive process. Not only does single-epoch training address both of these issues, but it has been proposed that they reduce the number of training samples required for successful training in both shallow [21] and deep networks [27, 28].

One of the earliest analytic neural network training algorithms was proposed in 2001 by Guo and Lyu [29]. Their proposed methodology trained a neural network in a layer-wise fashion, and resulted in a network with zero training loss. However, the required number of neurons in each layer was equal to the number of training samples. In 2006, Huang et al. [30] proposed extreme learning machines, which are deep neural networks where all the weights besides the last layer are randomly initialized. The optimal final layer weights can then be determined analytically using least squares. More recently, a methodology for layer-wise analytic training was presented by Zhuang et al. [31] who trained networks in two-layer modules by generating pseudo-labels from a random linear transformation of the final labels. In 2024, the authors further expanded their methodology to train convolutional neural networks [28] for pattern classification.

Although many propositions have been made for single-epoch network training, they do not provide guarantees of finding the global optima due to randomly initialized weights [30] or pseudo-labels [31, 28]. The single-epoch CNN training presented in [28], despite being the closest to the work presented in this thesis, is focused on pattern classification. The

extension to system theory is not obvious due to the lack of an explicit closed-form input-output expression.

1.2.3 Physics-Informed Neural Networks

Early implementations of neural networks for solving differential equations were presented by Dissanayake and Phan-Thien in 1994 [32] and Lagaris et al. in 1998 [33]. Despite this early work, it took advances in automatic differential and computational power for neural network differential equations solvers to gain mainstream appeal. In 2019, Raissi et al. [34] reintroduced the concept of using deep neural networks to solve differential equations, formally giving them the name of physics-informed neural networks (PINNs). PINNs extend the loss function to incorporate a penalty for deviating from the differential equations of the system. A useful general overview of training PINNs is presented in [35]. Although many extensions to PINNs have been proposed (e.g., sequential training [36], ensemble networks [37]), the 2023 work of Akhound-Sadegh et al. [38] on symmetries is the one extended to QNNs in this thesis. They proposed incorporating a loss term that penalizes deviations from the various symmetries of the differential equation.

Despite their versatility, one of the downsides of PINNs compared to other differential equations solvers, such as finite element methods (FEMs), is their slower training time due to the heavy computational demands of iterative training coupled with automatic differentiation [39]. In 2022, Desai et al. [40] proposed the usage of transfer learning to reduce the PINN training problem to optimizing the final layer. Furthermore, they showed that for linear differential equations, the final layer weights could be found in an analytic manner. In 2024, Zhou, Liu, et al. [41, 42] proposed using extreme learning machines for solving linear differential equation problems, specifically in the context of finding Lyapunov functions and iterative solving the Hamilton-Jacobi-Bellman equation for optimal control. By randomly selecting all the hidden layer weights, the final layer weights could be solved analytically. To the best of the author's knowledge, these are the only references that address single-epoch

PINN training.

Despite the importance of reducing training time, single-epoch PINN training methodologies are quite sparse, with the current propositions either requiring a pre-trained feature extractor [40] or the use of randomly initialized weights [41, 42]. Furthermore, incorporation of symmetries into the analytic weight formulation has not been explored to the best of the author's knowledge.

1.3 Contributions

This thesis addresses the least squares training of two-layer quadratic convolutional and physics-informed neural networks. The contributions of this work are listed below.

1. In Chapter 2, an alternate proof for the convex formulation of two-layer quadratic neural network (QNN) training is presented. Although this result was already shown in [12], the proof presented in this thesis considers the primal problem, whereas [12] uses duality.

2. Chapter 3 proposes the least squares formulation for the training of a two-layer convolutional quadratic neural network (CQNN), which uses a 2-norm loss function and no regularization. In doing so, insight is provided into the representational capacity of CQNNs by showing their equivalence to a subclass of QNNs. Examples of system identification and sensor fusion are used to show that CQNNs perform favorably relative to backpropagation trained CNNs, while having a significantly reduced training time.

3. Chapter 4 proposes the least squares formulation for the training of a two-layer quadratic physics-informed neural network (QPINN) with no regularization. Moreover, the required conditions and methodology for adding a symmetry loss term are provided. An example of the heat equation is used to show the performance of QPINNs in com-

parison to backpropagation-trained PINNs. Additionally, another example shows how QPINNs can be used to solve optimal control problems.

4. In Chapter 5, the optimal selection of quadratic activation function parameters to best approximate a ReLU activation function are presented for both minimizing the mean squared error and the maximum absolute error. Moreover, a proof that monomial lifting allows two-layer QNNs to act as universal approximators is provided. Finally, a methodology for implementing ensemble quadratic neural networks (EQNNs) is proposed and tested on a friction-induced vibration system.

1.4 Thesis Structure

This thesis is structured as follows. Chapter 2 provides the required preliminaries. This includes an overview of notation, of lifting for neural networks and system dynamics, and of quadratic neural networks (QNNs) and the formulation for their convex training.

Chapter 3 provides an introduction to two-layer convolutional quadratic neural networks (CQNNs), and proposes the least squares formulation for their training. The methodology is then applied to system identification and sensor fusion problems.

Chapter 4 provides an overview of physics-informed neural networks (PINNs), and proposes the least squares formulation for the training of two-layer quadratic physics-informed neural networks (QPINNs). The methodology is then applied to solve the heat equation and an optimal control problem.

Chapter 5 proposes an approach for optimal selection of quadratic activation function parameters, and provides two methods which increase the representational capacity of two-layer QNNs. Finally, Chapter 6 concludes the thesis with some closing remarks.

Chapter 2

Preliminaries

This chapter provides the relevant theoretical preliminaries for the work presented in this thesis. Section 2.1 outlines the notation used in this work. Section 2.2 introduces lifting as a motivation for quadratic networks. Section 2.3 contains the background on quadratic neural networks, along with the formulation for their convex training. Finally, Section 2.4 provides a new alternate proof for the convex training of quadratic neural networks introduced in Section 2.3.

2.1 Notation

Curly brackets are used to indicate indices. For example, $Z_{1_{\{i,j\}}}$ indicates the element at the i^{th} row and j^{th} column of matrix Z_1. Matrix and vector blocks are indicated by a colon. For example, $Z_{r_1:r_2,c_1:c_2}$ is the matrix constructed from rows r_1 to r_2 and columns c_1 to c_2 of matrix Z. $U_{n,m}$ and $O_{n,m}$ represent $n \times m$ matrices of ones and zeros respectively. The trace operator is written as $\mathbf{Tr}(\cdot)$, $Z \succeq 0$ and $Z \succ 0$ indicate that Z is positive semidefinite and positive definite, respectively, and the partial derivative operator of order k is written as $\frac{\partial^k}{\partial x_1 \cdots \partial x_k} = \partial_{x_1 \cdots x_k}$. Finally, $\partial\Omega$ is the boundary of the set Ω.

2.2 Lifting and Quadratic Dynamics

This section provides a brief overview of lifting with applications to neural networks and system dynamics. Additionally, it highlights some results from the realm of system theory that help motivate the study of quadratic neural networks.

2.2.1 Lifting

In the context of system theory, lifting is the process of augmenting a system's state to embed it in a higher-dimensional space with the goal of simplifying the dynamics in the lifted state space. Lifting is also relevant to neural networks, where the input vector may be lifted to simplify the input-output expression in the lifted feature space. Although applicable to both domains, this section will use the state nomenclature. Consider a state vector

$$x = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T \tag{2.1}$$

Lifting is performed by augmenting the state using M basis functions $\Phi_m : \mathbb{R}^n \to \mathbb{R}$, $m = 1, \ldots, M,\ M \geq 1$. The lifted state is given by

$$z = \begin{bmatrix} x_1 & x_2 & \cdots & x_n & \Phi_1(x) & \cdots & \Phi_M(x) \end{bmatrix}^T = \begin{bmatrix} z_1 & \cdots & z_{n+M} \end{bmatrix}^T \tag{2.2}$$

Example 1 (Lifted nonlinear pendulum)**.** *[25] Consider the state space representation of a damped nonlinear pendulum*

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\sin(x_1) - cx_2 \end{bmatrix}$$

where x_1 is the angle, x_2 is the angular velocity, and c is the damping coefficient. Using the basis functions $\Phi_1(x) = \sin(x_1)$, $\Phi_2(x) = \cos(x_1)$, the system can be lifted to quadratic

dynamics

$$z = \begin{bmatrix} x_1 \\ x_2 \\ \sin(x_1) \\ \cos(x_1) \end{bmatrix} \implies \dot{z} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_1 \cos(x_1) \\ -\dot{x}_1 \sin(x_1) \end{bmatrix} = \begin{bmatrix} x_2 \\ -\sin(x_1) - cx_2 \\ x_2 \cos(x_1) \\ -x_2 \sin(x_1) \end{bmatrix} = \begin{bmatrix} z_2 \\ -z_3 - cz_2 \\ z_2 z_4 \\ -z_2 z_3 \end{bmatrix}$$

with the added constraints $z_3 = \sin(z_1)$ *and* $z_4 = \cos(z_2)$.

2.2.2 Lifting to Quadratic Dynamics

It has been shown in references [23, 25, 43, 44] that any nonlinear dynamic system that is composed of elementary functions can be lifted to equivalent quadratic dynamics of the form

$$\dot{z} = A(z \otimes z) + Bz + C \tag{2.3}$$

where $\otimes$ is the Kronecker product defined as

$$V \otimes W = \begin{bmatrix} V_{1,1}W & V_{1,2}W & \cdots & V_{1,m}W \\ V_{2,1}W & V_{2,2}W & \cdots & V_{2,m}W \\ \vdots & \vdots & \ddots & \vdots \\ V_{n,1}W & V_{n,2}W & \cdots & V_{n,m}W \end{bmatrix} \in \mathbb{R}^{np \times mq} \tag{2.4}$$

with $V \in \mathbb{R}^{n \times m}$ and $W \in \mathbb{R}^{p \times q}$. An important point is that the lifted quadratic representation of the system will comprise of a finite set of states [23, 25]. This is in contrast to Koopman operator theory, which allows for lifting a system to a linear representation at the cost of potentially requiring an infinite number of states [45].

Example 2. *(Infinite linear versus finite quadratic) Consider the system*

$$\dot{x} = -x^3$$

Let us attempt to linearize it with lifting, starting with $z_1 = x$ and the lifted state $z_2 = x^3$. This gives $\dot{z}_2 = 3x^2\dot{x} = -3z_1^5$, which is still not linear. So, we introduce $z_3 = z_1^5$, which yields $\dot{z}_3 = 5z_1^4\dot{z}_1 = -5z_1^7$, which would require the introduction of another lifting variable. This process of introducing new lifting variables would continue indefinitely and result in an infinite number of lifted states required for the linear dynamics. Instead, using $z_1 = x$ and $z_2 = x^2$, the system can be lifted to quadratic dynamics in a finite state space

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} \dot{x} \\ 2x\dot{x} \end{bmatrix} = \begin{bmatrix} -x^3 \\ -2x^4 \end{bmatrix} = \begin{bmatrix} -z_1 z_2 \\ -2z_2^2 \end{bmatrix}$$

with the added constraint $z_2 = z_1^2$.

2.3 Two-Layer Quadratic Neural Networks

This section provides background on two-layer quadratic neural networks (QNNs), including their convex training. For simplicity, we shall only consider single output networks, as multiple outputs can be achieved by using multiple networks with the same inputs. The input-output expression for a two-layer, single output QNN, with m neurons in the hidden layer is given as

$$\hat{y}(x) = \sum_{j=1}^{m} \alpha_j \sigma(w_j^T x) \tag{2.5}$$

where $\sigma(z)$ is a second-order polynomial activation function, referred to as a quadratic activation function, defined as

$$\sigma(z) = az^2 + bz + c \tag{2.6}$$

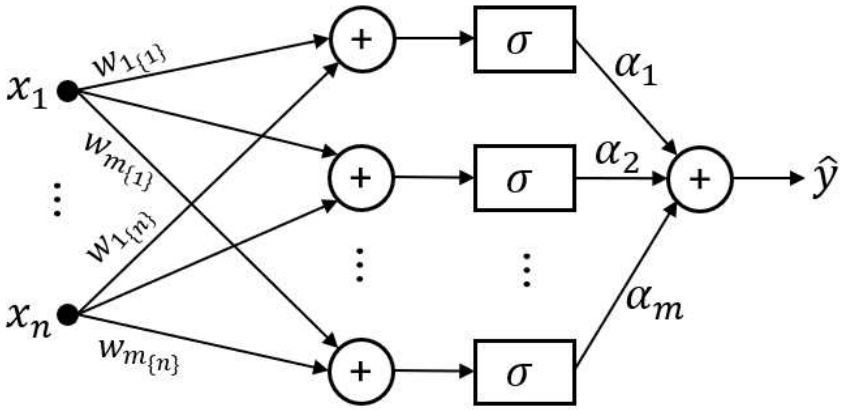


Figure 2.1: Single output quadratic neural network architecture

where $a \neq 0, b, c$ are constants that parametrize the activation function. Using the sample index i, $\hat{y}_i$ and y_i correspond to the network output and desired output (or label) for a given input vector $x_i \in \mathbb{R}^n$ containing n features. The first and second layer weights of the network for the j^{th} neuron are given by $w_j \in \mathbb{R}^n$ and α_j, respectively. The architecture of the network is shown in Figure 2.1. The training of a QNN defined by the input-output equation (2.5) is written as the following non-convex optimization problem in reference [12]

$$\begin{aligned} \min_{\{\alpha_j, w_j\}_{j=1}^m} \; & \mathcal{L}(\hat{y}, y) + \beta \sum_{j=1}^{m} |\alpha_j| \\ \text{s.t. } & \hat{y}_i = \sum_{j=1}^{m} \alpha_j \sigma(w_j^T x_i), \; i = 1, \ldots, N \\ & ||w_j||_2 = 1, \; j = 1, \ldots, m \end{aligned} \tag{2.7}$$

where $\mathcal{L}(\cdot)$ is a loss function which is applied to all network outputs and labels, and $\beta \geq 0$ is a parameter which modulates the L_1 regularization applied to the second layer weights.

2.3.1 Convex Training of QNNs

The convex formulation of QNN training was first shown in [12] through the use of duality. This thesis presents a new alternative proof of Lemma 1 in Section 2.4.

Lemma 1. *[12] For fixed $a \neq 0$, b, c, regularization term $\beta \geq 0$, and convex loss function $\mathcal{L}(\cdot)$, the non-convex problem (2.7) has the same global optimal solution as the convex problem*

$$\begin{aligned}
&\min_{Z^+, Z^-} \ \mathcal{L}(\hat{y}, y) + \beta(Z_4^+ + Z_4^-) \\
&\quad s.t. \ \hat{y}_i = \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} a(Z_1^+ - Z_1^-) & \frac{b}{2}(Z_2^+ - Z_2^-) \\ \frac{b}{2}(Z_2^+ - Z_2^-)^T & c(Z_4^+ - Z_4^-) \end{bmatrix} \begin{bmatrix} x_i \\ 1 \end{bmatrix}, \ i = 1, \ldots, N \\
&\quad Z_4^+ = \boldsymbol{Tr}(Z_1^+), \ Z_4^- = \boldsymbol{Tr}(Z_1^-) \\
&\quad Z^+ = \begin{bmatrix} Z_1^+ & Z_2^+ \\ (Z_2^+)^T & Z_4^+ \end{bmatrix}, \ Z^- = \begin{bmatrix} Z_1^- & Z_2^- \\ (Z_2^-)^T & Z_4^- \end{bmatrix} \\
&\quad Z^+ \succeq 0, \ Z^- \succeq 0
\end{aligned} \tag{2.8}$$

when the number of neurons $m \geq m^*$ where

$$m^* = rank(Z^{+*}) + rank(Z^{-*}) \tag{2.9}$$

where $Z^{+*}, Z^{-*} \in \mathbb{R}^{(n+1)\times(n+1)}$ are the solutions of the optimization problem (2.8).

The network weights $\{\alpha_j, w_j\}_{j=1}^m$ can be recovered from Z^+ and Z^- using the process of neural decomposition as presented in [12], which is based on Lemma 3 reviewed in Section 2.4.

2.3.2 Least Squares Training of QNNs

To formulate QNN training as a least squares problem, the LMI constraints of problem (2.8) must first be removed. It is shown in reference [21] that this is achievable at the cost of removing the regularization term and enforcing certain constraints on the a, b, c terms.

Lemma 2. *[21] If $\beta = 0$, $a > 0$, $c > 0$, and $b^2 - 4ac \geq 0$ then any solution of*

$$\begin{aligned} &\min_{Z_1, Z_2} \mathcal{L}(\hat{y}, y) \\ &\quad s.t.\ \hat{y}_i = \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} aZ_1 & \frac{b}{2}Z_2 \\ \frac{b}{2}(Z_2)^T & cZ_4 \end{bmatrix} \begin{bmatrix} x_i \\ 1 \end{bmatrix}, \quad i = 1, \ldots, N \\ &\quad\quad Z_4 = \boldsymbol{Tr}(Z_1) \end{aligned} \tag{2.10}$$

is also a solution of (2.8) where $Z_q = Z_q^+ - Z_q^-$, $q = 1, 2, 4$.

Remark 1 (Representational capacity of QNNs)**.** *The input-output expression for a QNN presented in problem (2.10) reveals that QNNs represent a subset of all quadratic forms due to the presence of the trace constraint. The existence of equivalent quadratic state dynamics for a sufficiently lifted state of any systems (as shown in Section 2.2) makes this a motivating factor for the application of QNNs to system theory.*

Example 3 (Invalid QNN)**.** *Let $a = b = c = 1$ and let $x \in \mathbb{R}$. The following expression*

$$\hat{y} = \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}$$

would not be a valid QNN, because with $Z_1 = 0$ and $Z_4 = 1$, then $\boldsymbol{Tr}(Z_1) \neq Z_4$.

Observing that the input-output expression in (2.10) is linear with respect to the elements of Z_1 and Z_2, the problem can be written in the following form

$$\begin{aligned} &\min_{\theta} \mathcal{L}(\hat{y}, y) \\ &\text{s.t. } \hat{y} = H\theta \end{aligned} \tag{2.11}$$

where the regressor matrix is given by

$$
\begin{aligned}
H &= \begin{bmatrix} aH_1 + cH_2 & bX \end{bmatrix}, \\
H_1 = \begin{bmatrix} \mathbf{vec}(x_1 x_1^T) \\ \vdots \\ \mathbf{vec}(x_N x_N^T) \end{bmatrix}, \quad H_2 &= \begin{bmatrix} U_{N,n} & O_{N,0.5n(n+1)-n} \end{bmatrix}, \quad X = \begin{bmatrix} x_1^T \\ \vdots \\ x_N^T \end{bmatrix}
\end{aligned} \tag{2.12}
$$

with $\mathbf{vec}(x_i x_i^T)$ being a row vector containing all of the upper triangular elements of $x_i x_i^T$

$$
\begin{aligned}
\mathbf{vec}(x_i x_i^T) &= \begin{bmatrix} diag_1 & diag_2 & \cdots & diag_n \end{bmatrix} \in \mathbb{R}^{1 \times 0.5n(n+1)} \\
diag_q &= \begin{bmatrix} x_{i_{\{1\}}} x_{i_{\{q\}}} & x_{i_{\{2\}}} x_{i_{\{q+1\}}} & \cdots & x_{i_{\{n-q+1\}}} x_{i_{\{n\}}} \end{bmatrix}
\end{aligned} \tag{2.13}
$$

and the weight vector is formed as follows

$$
\begin{aligned}
\theta &= \begin{bmatrix} \theta_{1,1} & 2\theta_{1,2} & \cdots & 2\theta_{1,n} & \theta_2 \end{bmatrix}^T \in \mathbb{R}^{0.5n(n+1)+n} \\
\theta_{1,i} = \begin{bmatrix} Z_{1_{\{1,i\}}} & Z_{1_{\{2,i+1\}}} & \cdots & Z_{1_{\{n-i+1,n\}}} \end{bmatrix}, \quad \theta_2 &= \begin{bmatrix} Z_{2_{\{1\}}} & Z_{2_{\{2\}}} & \cdots & Z_{2_{\{n\}}} \end{bmatrix}
\end{aligned} \tag{2.14}
$$

When using $\mathcal{L}(\hat{y}, y) = ||\hat{y} - y||_2^2$, problem (2.11) has the optimal solution given by

$$
\theta = (H^T H)^{-1} H^T y \tag{2.15}
$$

However, for the cases when $H^T H$ is not invertible, regularized least squares can be used, which changes the solution to

$$
\theta = (H^T H + \beta I)^{-1} H^T y \tag{2.16}
$$

It is important to note that the regularization term β present in (2.16) is not the same as the one present in the original training problem (2.7), thus the equivalence between problems only holds when $\beta = 0$. However, using small values of β will allow for an approximation of the solution of the problem (2.7), while preventing numerical issues.

2.4 Alternate Proof for Convex QNN Training

This section presents a new alternate proof for the convex training of QNNs (Lemma 1, Section 2.3). This proof requires the neural decomposition lemma from [12].

Lemma 3. *[12] Any rank-r, $r \geq 1$, matrix $Z \succeq 0$ where $\boldsymbol{Tr}(ZG) = 0$ with*

$$G = \begin{bmatrix} I & 0 \\ 0 & -1 \end{bmatrix}$$

has a decomposition $Z = \sum_{j=1}^{r} p_j p_j^T$ such that $p_j^T G p_j = 0,\ j = 1, \ldots, r$.

Starting from the non-convex QNN training problem (2.7), let $\alpha_j = \alpha_j^+ - \alpha_j^-$ with $\alpha_j^+ \geq 0$ and $\alpha_j^- \geq 0$. Then, the input-output expression from (2.7) can be rewritten as

$$\begin{aligned}
\hat{y}_i &= \sum_{j=1}^{m} \alpha_j \sigma(w_j^T x_i) = \sum_{j=1}^{m} \alpha_j (a x_i^T w_j w_j^T x_i + b w_j^T x_i + c) \\
&= \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} a \sum_{j=1}^{m} \alpha_j w_j w_j^T & \frac{b}{2} \sum_{j=1}^{m} \alpha_j w_j \\ \frac{b}{2} \sum_{j=1}^{m} \alpha_j w_j^T & c \sum_{j=1}^{m} \alpha_j \end{bmatrix} \begin{bmatrix} x_i \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} a \sum_{j=1}^{m} (\alpha_j^+ - \alpha_j^-) w_j w_j^T & \frac{b}{2} \sum_{j=1}^{m} (\alpha_j^+ - \alpha_j^-) w_j \\ \frac{b}{2} \sum_{j=1}^{m} (\alpha_j^+ - \alpha_j^-) w_j^T & c \sum_{j=1}^{m} (\alpha_j^+ - \alpha_j^-) \end{bmatrix} \begin{bmatrix} x_i \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} a(Z_1^+ - Z_1^-) & \frac{b}{2}(Z_2^+ - Z_2^-) \\ \frac{b}{2}(Z_2^+ - Z_2^-)^T & c(Z_4^+ - Z_4^-) \end{bmatrix} \begin{bmatrix} x_i \\ 1 \end{bmatrix}
\end{aligned} \tag{2.17}$$

where

$$\begin{aligned}
Z_1^+ &= \sum_{j=1}^{m} \alpha_j^+ w_j w_j^T, \quad Z_2^+ = \sum_{j=1}^{m} \alpha_j^+ w_j, \quad Z_4^+ = \sum_{j=1}^{m} \alpha_j^+ \\
Z_1^- &= \sum_{j=1}^{m} \alpha_j^- w_j w_j^T, \quad Z_2^- = \sum_{j=1}^{m} \alpha_j^- w_j, \quad Z_4^- = \sum_{j=1}^{m} \alpha_j^-
\end{aligned} \tag{2.18}$$

Without loss of generality

$$\alpha_j^+ = \begin{cases} |\alpha_j| + \epsilon_j & \alpha_j \geq 0 \\ \epsilon_j & \alpha_j < 0 \end{cases}, \quad \alpha_j^- = \begin{cases} |\alpha_j| + \epsilon_j & \alpha_j < 0 \\ \epsilon_j & \alpha_j \geq 0 \end{cases} \tag{2.19}$$

represent all possible $\alpha_j^+ \geq 0$ and $\alpha_j^- \geq 0$ which could form α_j, for $\epsilon_j \geq 0$. Furthermore, notice that the regularization loss of (2.7) is upper bounded,

$$\beta \sum_{j=1}^{m} |\alpha_j| = \beta \sum_{j=1}^{m} |\alpha_j^+ - \alpha_j^-| \leq \beta \sum_{j=1}^{m} (\alpha_j^+ + \alpha_j^-) \tag{2.20}$$

Using the new definition for α_j, consider the following problem with the same constraints as (2.7) but which uses the upper bound for its regularization loss term

$$\begin{aligned} \min_{\epsilon_j, \{w_j, \alpha_j^+, \alpha_j^-\}_{j=1}^m} \quad & \mathcal{L}(\hat{y}, y) + \beta(Z_4^+ + Z_4^-) \\ \text{s.t. } \hat{y}_i = & \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} a(Z_1^+ - Z_1^-) & \frac{b}{2}(Z_2^+ - Z_2^-) \\ \frac{b}{2}(Z_2^+ - Z_2^-)^T & c(Z_4^+ - Z_4^-) \end{bmatrix} \begin{bmatrix} x_i \\ 1 \end{bmatrix}, i = 1, \ldots, N \\ & ||w_j||_2 = 1, \quad \alpha_j = \alpha_j^+ - \alpha_j^-, \\ & \alpha_j^+ = \begin{cases} |\alpha_j| + \epsilon_j & \alpha_j \geq 0 \\ \epsilon_j & \alpha_j < 0 \end{cases}, \quad \alpha_j^- = \begin{cases} |\alpha_j| + \epsilon_j & \alpha_j < 0 \\ \epsilon_j & \alpha_j \geq 0 \end{cases}, \\ & \alpha_j^+ \geq 0, \quad \alpha_j^- \geq 0, \quad \epsilon_j \geq 0, \quad j = 1, \ldots, m \\ & Z_1^+ = \sum_{j=1}^{m} \alpha_j^+ w_j w_j^T, \quad Z_2^+ = \sum_{j=1}^{m} \alpha_j^+ w_j, \quad Z_4^+ = \sum_{j=1}^{m} \alpha_j^+ \\ & Z_1^- = \sum_{j=1}^{m} \alpha_j^- w_j w_j^T, \quad Z_2^- = \sum_{j=1}^{m} \alpha_j^- w_j, \quad Z_4^- = \sum_{j=1}^{m} \alpha_j^- \end{aligned} \tag{2.21}$$

If there is an optimal solution of problem (2.21) with $\alpha_j^+ = \alpha_j^{+,*}$, $\alpha_j^- = \alpha_j^{-,*}$, and $\epsilon_j = \epsilon_j^*$

(and therefore also $\alpha_j = \alpha_j^* = \alpha_j^{+,*} - \alpha_j^{-,*}$), the regularization term becomes

$$\begin{aligned}
\beta(Z_4^+ + Z_4^-) &= \beta \sum_{j=1}^{m} \left(\alpha_j^{+,*} + \alpha_j^{-,*}\right) = \beta \sum_{j \in \{j|\alpha_j^* \geq 0\}} \left(\alpha_j^{+,*} + \alpha_j^{-,*}\right) + \beta \sum_{j \in \{j|\alpha_j^* < 0\}} \left(\alpha_j^{+,*} + \alpha_j^{-,*}\right) \\
&= \beta \sum_{j \in \{j|\alpha_j^* \geq 0\}} \left((|\alpha_j^*| + \epsilon_j^*) + (\epsilon_j^*)\right) + \beta \sum_{j \in \{j|\alpha_j^* < 0\}} \left((\epsilon_j^*) + (|\alpha_j^*| + \epsilon_j^*)\right) \\
&= 2\beta \sum_{j=1}^{m} \epsilon_j^* + \beta \sum_{j=1}^{m} |\alpha_j^*|
\end{aligned} \tag{2.22}$$

When $\beta > 0$, (2.22) is minimized when $\epsilon_j^* = 0$. For cases where $\beta = 0$, $\epsilon_j^* = 0$ can also be selected without affecting the total loss. Thus, without loss of generality, $\epsilon_j^* = 0$ minimizes the loss term and results in a unique $\alpha_j^+ \geq 0$ and $\alpha_j^- \geq 0$ for each α_j according to (2.19). Furthermore, with $\epsilon_j = 0$, the optimal regularization term value is now the same as the one in problem (2.7). Therefore, the original non-convex problem (2.7) is equivalent to the problem

$$\begin{aligned}
\min_{\{w_j, \alpha_j^+, \alpha_j^-\}_{j=1}^m} \quad & \mathcal{L}(\hat{y}, y) + \beta(Z_4^+ + Z_4^-) \\
\text{s.t. } \hat{y}_i = & \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} a(Z_1^+ - Z_1^-) & \frac{b}{2}(Z_2^+ - Z_2^-) \\ \frac{b}{2}(Z_2^+ - Z_2^-)^T & c(Z_4^+ - Z_4^-) \end{bmatrix} \begin{bmatrix} x_i \\ 1 \end{bmatrix}, \quad i = 1, \ldots, N \\
& ||w_j||_2 = 1, \quad \alpha_j^+ \geq 0, \quad \alpha_j^- \geq 0, \quad j = 1, \ldots, m \\
& Z_1^+ = \sum_{j=1}^{m} \alpha_j^+ w_j w_j^T, \quad Z_2^+ = \sum_{j=1}^{m} \alpha_j^+ w_j, \quad Z_4^+ = \sum_{j=1}^{m} \alpha_j^+ \\
& Z_1^- = \sum_{j=1}^{m} \alpha_j^- w_j w_j^T, \quad Z_2^- = \sum_{j=1}^{m} \alpha_j^- w_j, \quad Z_4^- = \sum_{j=1}^{m} \alpha_j^-
\end{aligned} \tag{2.23}$$

We will now focus on the following set of constraints

$$\begin{aligned} ||w_j||_2 = 1, \quad \alpha_j^+ \geq 0, \quad \alpha_j^- \geq 0, \quad j = 1, \ldots, m \\ Z_1^+ = \sum_{j=1}^m \alpha_j^+ w_j w_j^T, \quad Z_2^+ = \sum_{j=1}^m \alpha_j^+ w_j, \quad Z_4^+ = \sum_{j=1}^m \alpha_j^+ \\ Z_1^- = \sum_{j=1}^m \alpha_j^- w_j w_j^T, \quad Z_2^- = \sum_{j=1}^m \alpha_j^- w_j, \quad Z_4^- = \sum_{j=1}^m \alpha_j^- \end{aligned} \tag{2.24}$$

and the following set of LMI and trace constraints

$$\begin{aligned} Z^+ = \begin{bmatrix} Z_1^+ & Z_2^+ \\ (Z_2^+)^T & Z_4^+ \end{bmatrix} \succeq 0, \quad Z_4^+ = \mathbf{Tr}(Z_1^+) \\ Z^- = \begin{bmatrix} Z_1^- & Z_2^- \\ (Z_2^-)^T & Z_4^- \end{bmatrix} \succeq 0, \quad Z_4^- = \mathbf{Tr}(Z_1^-) \end{aligned} \tag{2.25}$$

with the objective of showing (2.24)$\Longleftrightarrow$(2.25). First, we will shown that (2.24) $\Longrightarrow$ (2.25). The trace constraint of (2.25) results from

$$||w_j||_2 = 1 \implies \mathbf{Tr}\left(Z_1^+\right) = \mathbf{Tr}\left(\sum_{j=1}^m \alpha_j^+ w_j w_j^T\right) = \sum_{j=1}^m \alpha_j^+ w_j^T w_j = \sum_{j=1}^m \alpha_j^+ = Z_4^+ \tag{2.26}$$

To show the positive-semidefinite constraint of (2.25) we will show that $\eta^T Z^+ \eta \geq 0$ for all

real $\eta = [\eta_1^T \ \eta_2]^T$ where $\eta_1 \in \mathbb{R}^n$, $\eta_2 \in \mathbb{R}$

$$\begin{aligned}
\eta^T Z^+ \eta &= \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix}^T \begin{bmatrix} \sum_{j=1}^m \alpha_j^+ w_j w_j^T & \sum_{j=1}^m \alpha_j^+ w_j \\ \sum_{j=1}^m \alpha_j^+ w_j^T & \sum_{j=1}^m \alpha_j^+ \end{bmatrix} \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} \\
&= \sum_{j=1}^m \alpha_j^+ \left[\eta_1^T w_j w_j^T \eta_1 + \eta_1^T w_j \eta_2 + \eta_2 w_j^T \eta_1 + \eta_2^2\right] \\
&= \sum_{j=1}^m \alpha_j^+ \left[(\eta_1^T w_j)^2 + 2\eta_2(\eta_1^T w_j) + \eta_2^2\right] \\
&= \sum_{j=1}^m \alpha_j^+ \left[(\eta_1^T w_j) + \eta_2\right]^2 \\
\alpha_j^+ \geq 0, \ j = 1, \ldots, m \implies &\sum_{j=1}^m \alpha_j^+ \left[(\eta_1^T w_j) + \eta_2\right]^2 \geq 0 \implies Z^+ \succeq 0
\end{aligned} \tag{2.27}$$

The same can be shown for α_j^- and Z^-, thus (2.24) $\implies$ (2.25). We will now show that (2.25) $\implies$ (2.24) by following the neural decomposition lemma from [12]. Observe that Z^+ satisfies the required condition for Lemma 3, i.e.,

$$\begin{aligned}
\mathbf{Tr}(Z^+ G) &= \mathbf{Tr}\left(\begin{bmatrix} Z_1^+ & Z_2^+ \\ (Z_2^+)^T & Z_4^+ \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -1 \end{bmatrix}\right) = \mathbf{Tr}\left(\begin{bmatrix} Z_1^+ & -Z_2^+ \\ (Z_2^+)^T & -Z_4^+ \end{bmatrix}\right) \\
&= \mathbf{Tr}(Z_1^+) - Z_4^+ = 0 \iff \mathbf{Tr}(Z_1^+) = Z_4^+
\end{aligned} \tag{2.28}$$

Using Lemma 3, consider the neural decomposition of rank-r^+ matrix $Z^+ = \sum_{j=1}^{r^+} p_j p_j^T$ with $r^+ \geq 1$, where $p_j = [c_j^T \ d_j]^T$ with $c_j \in \mathbb{R}^n$ and $d_j \in \mathbb{R}$.

$$Z^+ = \sum_{j=1}^{r^+} p_j p_j^T = \sum_{j=1}^{r^+} \begin{bmatrix} c_j c_j^T & d_j c_j \\ d_j c_j^T & d_j^2 \end{bmatrix} = \begin{bmatrix} Z_1^+ & Z_2^+ \\ (Z_2^+)^T & Z_4^+ \end{bmatrix} \tag{2.29}$$

Observe that $p_j^T G p_j = 0$ implies $||c_j||_2^2 = d_j^2$. Now, we can make the following two assumptions. First, $||p_j||_2 > 0, \ j = 1, \ldots, r^+$, since if $||p_{j^*}||_2 = 0$ for some j^*, then we could remove $p_{j^*} p_{j^*}^T$ from the summation in (2.29) while leaving Z^+ invariant. Thus, $||c_j||_2^2 > 0, \ j =$

$1, \ldots, r^+$. Second, $d_j \geq 0,\ j = 1, \ldots, r^+$, since if $d_{j^*} < 0$ for some j^*, redefine $p_{j^*} = -p_{j^*}$, which does not affect the decomposition nor the equality $p_{j^*}^T G p_{j^*} = 0$. Following these observations and assumptions, letting $u_j = \frac{c_j}{||c_j||_2}$, the following holds.

$$\begin{aligned} Z_1^+ &= \sum_{j=1}^{r^+} c_j c_j^T = \sum_{j=1}^{r^+} u_j u_j^T ||c_j||_2^2 = \sum_{j=1}^{r^+} d_j^2 u_j u_j^T \\ Z_2^+ &= \sum_{j=1}^{r^+} d_j c_j = \sum_{j=1}^{r^+} d_j u_j ||c_j||_2 = \sum_{j=1}^{r^+} d_j^2 u_j \\ Z_4^+ &= \sum_{j=1}^{r^+} d_j^2 \end{aligned} \tag{2.30}$$

Perform the same neural decomposition procedure on the rank-r^- matrix $Z^- = \sum_{j=1+r^+}^{r^+ + r^-} p_j p_j^T$ with $r^- \geq 1$. Therefore, letting

$$\begin{aligned} &w_j = u_j, \quad \alpha_j^+ = d_j^2, \quad \alpha_j^- = 0, \quad j = 1, \ldots, r^+ \\ &w_j = u_j, \quad \alpha_j^+ = 0, \quad \alpha_j^- = d_j^2, \quad j = r^+ + 1, \ldots, r^+ + r^- \end{aligned} \tag{2.31}$$

it is proven that (2.25) $\implies$ (2.24) when $r^+ \geq 1$ and $r^- \geq 0$, because $||w_j||_2^2 = ||u_j||_2^2 = 1$, $\alpha_j^+ \geq 0$, and $\alpha_j^- \geq 0$ so long as the number of hidden layer neurons m satisfies

$$m \geq r^+ + r^- = \text{rank}(Z^+) + \text{rank}(Z^-) \tag{2.32}$$

For the special case of $r^+ = 0$ or $r^- = 0$, the neural decomposition of the respective Z matrix can be ignored and (2.25) $\implies$ (2.24) will still hold. Thus (2.25)$\Longleftrightarrow$(2.24), which means that problem (2.23) (and therefore the non-convex training problem (2.7)) is equivalent to the

problem

$$
\begin{aligned}
&\min_{Z^+,Z^-} \ \mathcal{L}(\hat{y},y)+\beta(Z_4^+ + Z_4^-) \\
&\quad \text{s.t. } \hat{y}_i = \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} a(Z_1^+ - Z_1^-) & \frac{b}{2}(Z_2^+ - Z_2^-) \\ \frac{b}{2}(Z_2^+ - Z_2^-)^T & c(Z_4^+ - Z_4^-) \end{bmatrix} \begin{bmatrix} x_i \\ 1 \end{bmatrix}, \quad i=1,\dots,N \\
&\qquad Z_4^+ = \mathbf{Tr}(Z_1^+),\ Z_4^- = \mathbf{Tr}(Z_1^-) \\
&\qquad Z^+ = \begin{bmatrix} Z_1^+ & Z_2^+ \\ (Z_2^+)^T & Z_4^+ \end{bmatrix},\ Z^- = \begin{bmatrix} Z_1^- & Z_2^- \\ (Z_2^-)^T & Z_4^- \end{bmatrix} \\
&\qquad Z^+ \succeq 0,\ Z^- \succeq 0
\end{aligned}
\tag{2.33}
$$

when the number of neurons $m \geq m^*$ where

$$
m^* = \mathbf{Rank}(Z^{+,*}) + \mathbf{Rank}(Z^{-,*}) \tag{2.34}
$$

with $Z^{+,*}, Z^{-,*} \in \mathbb{R}^{(n+1)\times(n+1)}$ being the solutions of the optimization problem (2.33). The proof is thus complete.

Chapter 3

Least Squares Training of Convolutional Quadratic Neural Networks

This chapter proposes a methodology to formulate the training of a two-layer convolutional quadratic neural network (CQNN) as a least squares problem. In addition to an analytic solution for the optimal weights of the network, the proposed methodology also provides insight into the representational capacity of CQNNs by showing their equivalence to a subclass of quadratic neural networks (QNNs). The chapter is structured as follows. Section 3.1 provides the preliminaries related to CQNNs and their convex training. Section 3.2 shows how CQNN training can be formulated as a QNN training problem. Section 3.3 presents least squares training of a CQNN. Section 3.4 extends the main result to multi-channel filtering. Finally, Section 3.5 presents examples comparing CQNNs to backpropagation-trained convolutional neural networks.

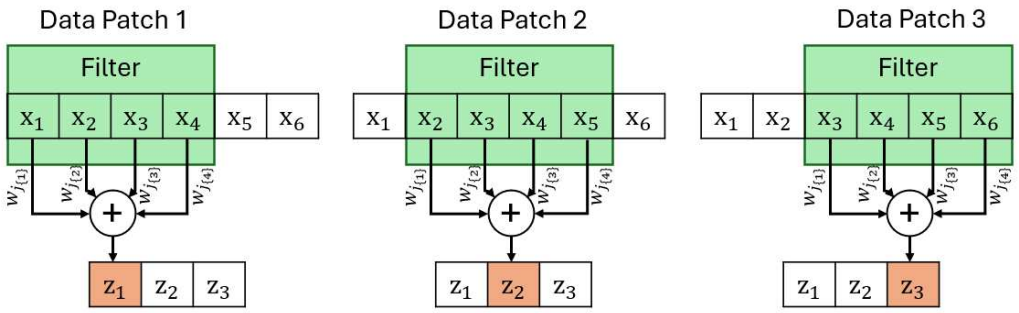


Figure 3.1: Visualization of convolutional layer feature extraction, filter of length $f = 4$ and stride of one passing over data vector of length 6

3.1 Convolutional Quadratic Neural Networks

Convolutional neural networks (CNNs) are neural networks that are specialized in handling sequential data such as images or time-series signals. They employ convolutional layers that use filters (or kernels) to perform feature extraction over different patches of input data. This process is analogous to sliding the filter over the input vector as shown in Figure 3.1. The filter length f dictates the length of the data patches covered by the filter and is the primary tuning parameter for convolutional layers. Secondary parameters for convolutional layers are stride, padding, and dilation. However, for simplicity, this thesis will consider a stride length of 1, no padding, and no dilation. The data patch concept is key to the theoretical work on convolutional quadratic neural networks (CQNNs) presented in this thesis. Consider an input vector of length n

$$x = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T \tag{3.1}$$

from which K patches of data are formed such that each patch χ_k contains f sequential features of x. Since the last element of χ_K is also the last element of x, $K = n - f + 1$.

$$\begin{aligned} \chi &= \begin{bmatrix} \chi_1 & \chi_2 & \cdots & \chi_k & \cdots & \chi_K \end{bmatrix}^T \\ \chi_k &= \begin{bmatrix} x_k & x_{k+1} & \cdots & x_{k+f-1} \end{bmatrix}^T = x_{k:k+f-1} \end{aligned} \tag{3.2}$$

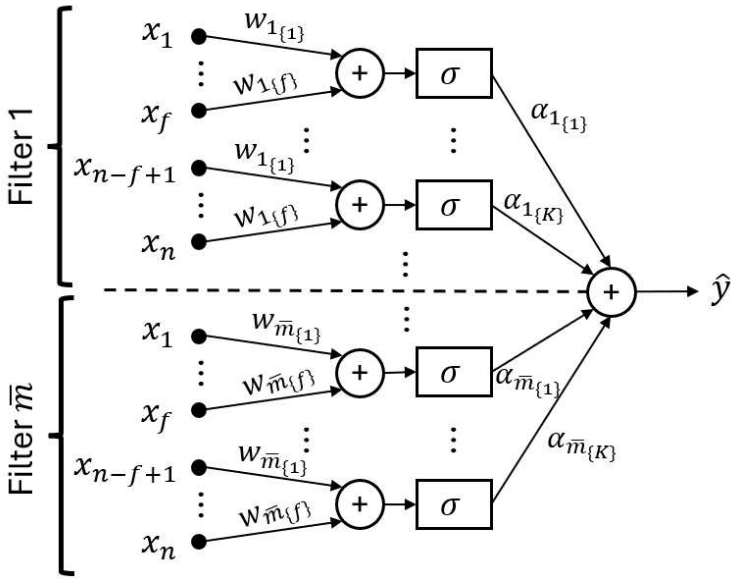


Figure 3.2: Single output convolutional quadratic neural network architecture

Example 4 (Data patches)**.** *Consider an input vector* $x = [x_1\ x_2\ x_3\ x_4\ x_5]^T$*, a filter with* $f = 3$ *would produce the following data patches* $\chi_1 = [x_1\ x_2\ x_3]^T$*,* $\chi_2 = [x_2\ x_3\ x_4]^T$*, and* $\chi_3 = [x_3\ x_4\ x_5]^T$*, for a total of* $K = n - f + 1 = 3$ *data patches with* $n = 5$*.*

Using this formal data patch definition, equation (3.3) represents the input-output expression of a single output, two-layer CQNN using $\bar{m}$ filters of length f, and quadratic activation function $\sigma(z)$ defined by (2.6). The architecture for a CQNN is shown in Figure 3.2.

$$\hat{y}(x) = \sum_{j=1}^{\bar{m}} \sum_{k=1}^{K} \alpha_{j_{\{k\}}} \sigma(w_j^T \chi_k) \tag{3.3}$$

The training of the CQNN defined by input-output equation (3.3) is written as the following non-convex optimization problem in [12], where $\alpha_j = [\alpha_{j_{\{1\}}}\ \cdots\ \alpha_{j_{\{K\}}}]^T$,

$$\begin{aligned} \min_{\{\alpha_j, w_j\}_{j=1}^{\bar{m}}} \ & \mathcal{L}(\hat{y}, y) + \beta \sum_{j=1}^{\bar{m}} ||\alpha_j||_1 \\ \text{s.t. } & \hat{y}_i = \sum_{j=1}^{\bar{m}} \sum_{k=1}^{K} \alpha_{j_{\{k\}}} \sigma(w_j^T \chi_{i_{\{k\}}}),\ i = 1, \ldots, N \\ & ||w_j||_2 = 1,\ j = 1, \ldots, \bar{m} \end{aligned} \tag{3.4}$$

3.1.1 Convex Training of CQNNs

The convex formulation of CQNN training was first shown in [12] through the use of duality. However, it will later be seen in Theorem 1 that CQNNs are a subclass of QNNs, therefore the alternate proof of Section 2.4 could also be modified to prove the following Lemma.

Lemma 4. *[12] For fixed $a \neq 0$, b, c, regularization term $\beta \geq 0$, and convex loss function $\mathcal{L}(\cdot)$, the non-convex problem (3.4) has the same global optimal solution as the convex problem*

$$\begin{aligned}
&\min_{\{Z^{+,k},Z^{-,k}\}_{k=1}^{K}} \mathcal{L}(\hat{y},y) + \beta \sum_{k=1}^{K} (Z_4^{+,k} + Z_4^{-,k}) \\
&s.t.\ \hat{y}_i = \sum_{k=1}^{K} \left(\begin{bmatrix} \chi_{i_{\{k\}}} \\ 1 \end{bmatrix}^T \begin{bmatrix} a(Z_1^{+,k} - Z_1^{-,k}) & \frac{b}{2}(Z_2^{+,k} - Z_2^{-,k}) \\ \frac{b}{2}(Z_2^{+,k} - Z_2^{-,k})^T & c(Z_4^{+,k} - Z_4^{-,k}) \end{bmatrix} \begin{bmatrix} \chi_{i_{\{k\}}} \\ 1 \end{bmatrix} \right), \\
&Z_4^{+,k} = \boldsymbol{Tr}(Z_1^{+,k}),\ Z_4^{-,k} = \boldsymbol{Tr}(Z_1^{-,k}) \\
&Z^{+,k} = \begin{bmatrix} Z_1^{+,k} & Z_2^{+,k} \\ (Z_2^{+,k})^T & Z_4^{+,k} \end{bmatrix},\ Z^{-,k} = \begin{bmatrix} Z_1^{-,k} & Z_2^{-,k} \\ (Z_2^{-,k})^T & Z_4^{-,k} \end{bmatrix} \\
&Z^{+,k} \succeq 0,\ Z^{-,k} \succeq 0 \\
&k = 1, \ldots, K, \quad i = 1, \ldots, N
\end{aligned} \tag{3.5}$$

when the number of filters $\bar{m} \geq \bar{m}^*$ where

$$\bar{m}^* = K \sum_{k=1}^{K} \left(rank(Z^{+,k*}) + rank(Z^{-,k*}) \right) \tag{3.6}$$

and where $Z^{+,k*}, Z^{-,k*} \in \mathbb{R}^{(f+1)\times(f+1)}$ for $k = 1, \ldots, K$ are the solutions of the optimization problem (3.5).

Similarly to the QNNs, the network weights $\{\alpha_j, w_j\}_{j=1}^{\bar{m}}$ can be recovered from $Z^{+,k}$ and $Z^{-,k}$ using the process of neural decomposition presented in [12].

Remark 2 (Design parameter $\bar{m}$)**.** *Using the convex CQNN training formulation in (3.5), the number of filters $\bar{m}$ is no longer a design parameter, instead it is a by-product of the training and the neural decomposition process.*

3.2 CQNN Formulated as a QNN

This section shows how CQNN training can be formulated as an equivalent QNN training problem with added constraints. The following lemmas will be required for the main theorem of this section.

Lemma 5. *For $a > 0$, $b \neq 0$, and $c > 0$, any real symmetric matrix Z of the form*

$$Z = \begin{bmatrix} aZ_1 & \frac{b}{2}Z_2 \\ \frac{b}{2}(Z_2)^T & c\boldsymbol{Tr}(Z_1) \end{bmatrix} \in \mathbb{R}^{n+1 \times n+1} \tag{3.7}$$

has a decomposition, $Z = Z^+ - Z^-$, such that $Z^+ \succeq 0$ and $Z^- \succeq 0$ where

$$Z^+ = \begin{bmatrix} aZ_1^+ & \frac{b}{2}Z_2^+ \\ \frac{b}{2}(Z_2^+)^T & c\boldsymbol{Tr}(Z_1^+) \end{bmatrix}, \quad Z^- = \begin{bmatrix} aZ_1^- & \frac{b}{2}Z_2^- \\ \frac{b}{2}(Z_2^-)^T & c\boldsymbol{Tr}(Z_1^-) \end{bmatrix} \tag{3.8}$$

Proof. Since Z is a real symmetric matrix, it can be written as the difference between two positive semidefinite (PSD) matrices $Z = \tilde{Z}^+ - \tilde{Z}^-$ ([46], page 119). Let

$$\tilde{Z}^+ = \begin{bmatrix} A_1^+ & A_2^+ \\ (A_2^+)^T & A_4^+ \end{bmatrix} \succeq 0, \quad \tilde{Z}^- = \begin{bmatrix} A_1^- & A_2^- \\ (A_2^-)^T & A_4^- \end{bmatrix} \succeq 0$$

where A_1^+ and A_1^- are $n \times n$ matrices, A_2^+ and A_2^- are vectors of length n, and A_4^+ and A_4^- are scalars. Since $a > 0$, $b \neq 0$, $c > 0$, without loss of generality, we can rewrite $\tilde{Z}^+$ and $\tilde{Z}^-$ as

$$\tilde{Z}^+ = \begin{bmatrix} a\tilde{Z}_1^+ & \frac{b}{2}\tilde{Z}_2^+ \\ \frac{b}{2}(\tilde{Z}_2^+)^T & c\tilde{Z}_4^+ \end{bmatrix} \succeq 0, \quad \tilde{Z}^- = \begin{bmatrix} a\tilde{Z}_1^- & \frac{b}{2}\tilde{Z}_2^- \\ \frac{b}{2}(\tilde{Z}_2^-)^T & c\tilde{Z}_4^- \end{bmatrix} \succeq 0$$

where

$$\tilde{Z}_1^+ = a^{-1}A_1^+, \quad \tilde{Z}_2^+ = 2b^{-1}A_2^+, \quad \tilde{Z}_4^+ = c^{-1}A_4^+$$
$$\tilde{Z}_1^- = a^{-1}A_1^-, \quad \tilde{Z}_2^- = 2b^{-1}A_2^-, \quad \tilde{Z}_4^- = c^{-1}A_4^-$$

Furthermore, without loss of generality, for any $K \in \mathbb{R}^{n+1\times n+1}$, we can write

$$\begin{aligned} Z &= \tilde{Z}^+ - \tilde{Z}^- + K - K \\ &= (\tilde{Z}^+ + K) - (\tilde{Z}^- + K) \\ &= Z^+ - Z^- \end{aligned}$$

where $Z^+ = \tilde{Z}^+ + K$ and $Z^- = \tilde{Z}^- + K$. Notice that if $K \succeq 0$, then $Z^+ \succeq 0$ and $Z^- \succeq 0$, as the summation of two PSD matrices is also PSD. Additionally, from the structure of Z in (3.7), we know that

$$\mathbf{Tr}(\tilde{Z}_1^+ - \tilde{Z}_1^-) = \tilde{Z}_4^+ - \tilde{Z}_4^- \iff \mathbf{Tr}(\tilde{Z}_1^+) - \tilde{Z}_4^+ = \mathbf{Tr}(\tilde{Z}_1^-) - \tilde{Z}_4^- \tag{3.9}$$

This will be used later in the proof. There are two cases to consider:

1. Case 1: $\mathbf{Tr}(\tilde{Z}_1^+) - \tilde{Z}_4^+ \geq 0$

   Define $k = \mathbf{Tr}(\tilde{Z}_1^+) - \tilde{Z}_4^+ \geq 0$ and K as

   $$K = \begin{bmatrix} O_{n,n} & O_{n,1} \\ O_{1,n} & ck \end{bmatrix}$$

   Since $ck \geq 0$ it follows that $K \succeq 0$, and therefore $Z^+ = \tilde{Z}^+ + K \succeq 0$ and $Z^- = \tilde{Z}^- + K \succeq 0$. Computing $Z^+ = \tilde{Z}^+ + K$ yields

   $$Z^+ = \begin{bmatrix} a\tilde{Z}_1^+ & \frac{b}{2}\tilde{Z}_2^+ \\ \frac{b}{2}(\tilde{Z}_2^+)^T & c(\tilde{Z}_4^+ + k) \end{bmatrix} = \begin{bmatrix} a\tilde{Z}_1^+ & \frac{b}{2}\tilde{Z}_2^+ \\ \frac{b}{2}(\tilde{Z}_2^+)^T & c\mathbf{Tr}(\tilde{Z}_1^+) \end{bmatrix}$$

   Computing $Z^- = \tilde{Z}^- + K$ gives

   $$Z^- = \begin{bmatrix} a\tilde{Z}_1^- & \frac{b}{2}\tilde{Z}_2^- \\ \frac{b}{2}(\tilde{Z}_2^-)^T & c(\tilde{Z}_4^- + k) \end{bmatrix} = \begin{bmatrix} a\tilde{Z}_1^- & \frac{b}{2}\tilde{Z}_2^- \\ \frac{b}{2}(\tilde{Z}_2^-)^T & c(\tilde{Z}_4^- + \mathbf{Tr}(\tilde{Z}_1^+) - \tilde{Z}_4^+) \end{bmatrix}$$

Using (3.9) yields

$$Z^- = \begin{bmatrix} a\tilde{Z}_1^- & \frac{b}{2}\tilde{Z}_2^- \\ \frac{b}{2}(\tilde{Z}_2^-)^T & c(\tilde{Z}_4^- + \mathbf{Tr}(\tilde{Z}_1^-) - \tilde{Z}_4^-) \end{bmatrix} = \begin{bmatrix} a\tilde{Z}_1^- & \frac{b}{2}\tilde{Z}_2^- \\ \frac{b}{2}(\tilde{Z}_2^-)^T & c\mathbf{Tr}(\tilde{Z}_1^-) \end{bmatrix}$$

Therefore, $Z^+ \succeq 0$ and $Z^- \succeq 0$ can be written as

$$Z^+ = \begin{bmatrix} aZ_1^+ & \frac{b}{2}Z_2^+ \\ \frac{b}{2}(Z_2^+)^T & c\mathbf{Tr}(Z_1^+) \end{bmatrix}, \quad Z^- = \begin{bmatrix} aZ_1^- & \frac{b}{2}Z_2^- \\ \frac{b}{2}(Z_2^-)^T & c\mathbf{Tr}(Z_1^-) \end{bmatrix}$$

where $Z_1^+ = \tilde{Z}_1^+$, $Z_1^- = \tilde{Z}_1^-$, $Z_2^+ = \tilde{Z}_2^+$, and $Z_2^- = \tilde{Z}_2^-$.

2. Case 2: $\mathbf{Tr}(\tilde{Z}_1^+) - \tilde{Z}_4^+ < 0$

   Define $k = \tilde{Z}_4^+ - \mathbf{Tr}(\tilde{Z}_1^+) > 0$ and K as

$$K = \begin{bmatrix} an^{-1}kI_{n,n} & O_{n,1} \\ O_{1,n} & 0 \end{bmatrix}$$

   where $I_{n,n}$ is an $n \times n$ identity matrix. Additionally, since $an^{-1}k > 0$ it follows that $an^{-1}kI_{n,n} \succ 0$, and thus $K \succeq 0$, which implies $Z^+ = \tilde{Z}^+ + K \succeq 0$ and $Z^- = \tilde{Z}^- + K \succeq 0$. Computing $Z^+ = \tilde{Z}^+ + K$ yields

$$Z^+ = \begin{bmatrix} a(\tilde{Z}_1^+ + n^{-1}kI_{n,n}) & \frac{b}{2}\tilde{Z}_2^+ \\ \frac{b}{2}(\tilde{Z}_2^+)^T & c\tilde{Z}_4^+ \end{bmatrix} = \begin{bmatrix} a(\tilde{Z}_1^+ + n^{-1}(\tilde{Z}_4^+ - \mathbf{Tr}(\tilde{Z}_1^+))I_{n,n}) & \frac{b}{2}\tilde{Z}_2^+ \\ \frac{b}{2}(\tilde{Z}_2^+)^T & c\tilde{Z}_4^+ \end{bmatrix}$$

   Taking the trace of the upper left sub-matrix yields

$$\begin{aligned} \mathbf{Tr}\left(\tilde{Z}_1^+ + n^{-1}(\tilde{Z}_4^+ - \mathbf{Tr}(\tilde{Z}_1^+))I_{n,n}\right) &= \mathbf{Tr}(\tilde{Z}_1^+) + n^{-1}(\tilde{Z}_4^+ - \mathbf{Tr}(\tilde{Z}_1^+))\mathbf{Tr}(I_{n,n}) \\ &= \mathbf{Tr}(\tilde{Z}_1^+) + \tilde{Z}_4^+ - \mathbf{Tr}(\tilde{Z}_1^+) \\ &= \tilde{Z}_4^+ \end{aligned}$$

Computing $Z^- = \tilde{Z}^- + K$ yields

$$Z^- = \begin{bmatrix} a(\tilde{Z}_1^- + n^{-1}kI_{n,n}) & \frac{b}{2}\tilde{Z}_2^- \\ \frac{b}{2}(\tilde{Z}_2^-)^T & c(\tilde{Z}_4^-) \end{bmatrix} = \begin{bmatrix} a(\tilde{Z}_1^- + n^{-1}(\tilde{Z}_4^+ - \mathbf{Tr}(\tilde{Z}_1^+))I_{n,n}) & \frac{b}{2}\tilde{Z}_2^- \\ \frac{b}{2}(\tilde{Z}_2^-)^T & c(\tilde{Z}_4^-) \end{bmatrix}$$

Using (3.9) yields

$$Z^- = \begin{bmatrix} a(\tilde{Z}_1^- + n^{-1}(\tilde{Z}_4^- - \mathbf{Tr}(\tilde{Z}_1^-))I_{n,n}) & \frac{b}{2}\tilde{Z}_2^- \\ \frac{b}{2}(\tilde{Z}_2^-)^T & c(\tilde{Z}_4^-) \end{bmatrix}$$

Taking the trace of the upper left sub-matrix yields

$$\begin{aligned} \mathbf{Tr}\left(\tilde{Z}_1^- + n^{-1}(\tilde{Z}_4^- - \mathbf{Tr}(\tilde{Z}_1^-))I_{n,n}\right) &= \mathbf{Tr}(\tilde{Z}_1^-) + n^{-1}(\tilde{Z}_4^- - \mathbf{Tr}(\tilde{Z}_1^-))\mathbf{Tr}(I_{n,n}) \\ &= \mathbf{Tr}(\tilde{Z}_1^-) + \tilde{Z}_4^- - \mathbf{Tr}(\tilde{Z}_1^-) \\ &= \tilde{Z}_4^- \end{aligned}$$

Therefore, $Z^+ \succeq 0$ and $Z^- \succeq 0$ can be written as

$$Z^+ = \begin{bmatrix} aZ_1^+ & \frac{b}{2}Z_2^+ \\ \frac{b}{2}(Z_2^+)^T & c\mathbf{Tr}(Z_1^+) \end{bmatrix}, \quad Z^- = \begin{bmatrix} aZ_1^- & \frac{b}{2}Z_2^- \\ \frac{b}{2}(Z_2^-)^T & c\mathbf{Tr}(Z_1^-) \end{bmatrix}$$

where $Z_1^+ = \tilde{Z}_1^+ + n^{-1}(\tilde{Z}_4^+ - \mathbf{Tr}(\tilde{Z}_1^+))I_{n,n}$, $\quad Z_1^- = \tilde{Z}_1^- + n^{-1}(\tilde{Z}_4^- - \mathbf{Tr}(\tilde{Z}_1^-))I_{n,n}$, $Z_2^+ = \tilde{Z}_2^+$, and $Z_2^- = \tilde{Z}_2^-$.

Thus, the matrix Z can be decomposed as

$$Z = Z^+ - Z^- = \begin{bmatrix} aZ_1^+ & \frac{b}{2}Z_2^+ \\ \frac{b}{2}(Z_2^+)^T & c\mathbf{Tr}(Z_1^+) \end{bmatrix} - \begin{bmatrix} aZ_1^- & \frac{b}{2}Z_2^- \\ \frac{b}{2}(Z_2^-)^T & c\mathbf{Tr}(Z_1^-) \end{bmatrix}$$

where $Z^+ \succeq 0$ and $Z^- \succeq 0$. □

The following Lemma is taken from the main theorem of [21], but is stated on its own as it will be used multiple times throughout the thesis.

Lemma 6. *[21] For $a > 0$, $c > 0$, and $b^2 - 4ac \geq 0$, for any real symmetric matrix Z of the form*

$$Z = \begin{bmatrix} aZ_1 & \frac{b}{2}Z_2 \\ \frac{b}{2}(Z_2)^T & c\boldsymbol{Tr}(Z_1) \end{bmatrix} \tag{3.10}$$

if $Z \succeq 0$, then

$$Z' = \begin{bmatrix} Z_1 & Z_2 \\ (Z_2)^T & \boldsymbol{Tr}(Z_1) \end{bmatrix} \succeq 0 \tag{3.11}$$

Proof. Using Schur's complement, $Z \succeq 0$ is equivalent to

$$\begin{aligned} c\mathbf{Tr}(Z_1) &\geq 0 \\ \left(1 - \mathbf{Tr}(Z_1)\mathbf{Tr}^\dagger(Z_1^+)\right)\frac{b}{2}(Z_2)^T &= 0 \\ aZ_1 - \frac{b^2}{4}Z_2\mathbf{Tr}^\dagger(cZ_1)(Z_2)^T &\succeq 0 \end{aligned} \tag{3.12}$$

where, defining $t = \mathbf{Tr}(cZ_1)$,

$$\mathbf{Tr}^\dagger(cZ_1) = \begin{cases} 0 & t = 0 \\ \mathbf{Tr}^{-1}(cZ_1) & t \neq 0 \end{cases}$$

is the Moore-Penrose pseudo-inverse of $\mathbf{Tr}(cZ_1)$. When $t = 0$, the conditions in (3.12) become $Z_2 = 0$, $Z_1 \succeq 0$, for which $Z' \succeq 0$. When $t \neq 0$ and $c > 0$, the conditions in (3.12) become

$$\frac{4ac}{b^2}Z_1\mathbf{Tr}(Z_1) \succeq Z_2(Z_2)^T$$

which by Schur's complement implies $Z' \succeq 0$ if $b^2 - 4ac \geq 0$ and $ac > 0$. □

Starting from the convex formulation of CQNN training presented in problem (3.5), the following lemma is an extension of its QNN counterpart (Section 2.3, Lemma 2 [21]) and will be used to remove the LMI constraints of problem (3.5).

Lemma 7. *If $\beta = 0$, $a > 0$, $c > 0$, and $b^2 - 4ac \geq 0$ then any solution of*

$$\begin{aligned} &\min_{\{Z_1^k, Z_2^k\}_{k=1}^K} \mathcal{L}(\hat{y}, y) \\ &s.t.\ \hat{y}_i = \sum_{k=1}^{K} \left(\begin{bmatrix} \chi_{i_{\{k\}}} \\ 1 \end{bmatrix}^T \begin{bmatrix} aZ_1^k & \frac{b}{2} Z_2^k \\ \frac{b}{2}(Z_2^k)^T & cZ_4^k \end{bmatrix} \begin{bmatrix} \chi_{i_{\{k\}}} \\ 1 \end{bmatrix} \right),\ i = 1, \ldots, N \\ &Z_4^k = \boldsymbol{Tr}(Z_1^k) \end{aligned} \tag{3.13}$$

is also a solution of (3.5) where $Z_q^k = Z_q^{+,k} - Z_q^{-,k}$, $q = 1, 2, 4$.

Proof. The proof is the same for all samples, so the index i is dropped. Let there be a solution to (3.13) with $a > 0$, $c > 0$, $b^2 - 4ac \geq 0$. Consider the matrix for the k^{th} data patch

$$Z^k = \begin{bmatrix} aZ_1^k & \frac{b}{2} Z_2^k \\ \frac{b}{2}(Z_2^k)^T & cZ_4^k \end{bmatrix} = \begin{bmatrix} aZ_1^k & \frac{b}{2} Z_2^k \\ \frac{b}{2}(Z_2^k)^T & c\mathbf{Tr}(Z_1^k) \end{bmatrix}$$

where the constraint $Z_4^k = \mathbf{Tr}(Z_1^k)$ from (3.13) has been used. Since Z^k is real symmetric, invoking Lemma 5, it can be written as the difference of two positive semidefinite matrices $Z^k = Z^{+,k} - Z^{-,k}$, where

$$Z^{+,k} = \begin{bmatrix} aZ_1^{+,k} & \frac{b}{2} Z_2^{+,k} \\ \frac{b}{2}(Z_2^{+,k})^T & c\mathbf{Tr}(Z_1^{+,k}) \end{bmatrix} \succeq 0, \quad Z^{-,k} = \begin{bmatrix} aZ_1^{-,k} & \frac{b}{2} Z_2^{-,k} \\ \frac{b}{2}(Z_2^{-,k})^T & c\mathbf{Tr}(Z_1^{-,k}) \end{bmatrix} \succeq 0$$

Invoking Lemma 6, it can be seen that

$$\begin{bmatrix} aZ_1^{+,k} & \frac{b}{2} Z_2^{+,k} \\ \frac{b}{2}(Z_2^{+,k})^T & c\mathbf{Tr}(Z_1^{+,k}) \end{bmatrix} \succeq 0 \implies \begin{bmatrix} Z_1^{+,k} & Z_2^{+,k} \\ (Z_2^{+,k})^T & \mathbf{Tr}(Z_1^{+,k}) \end{bmatrix} \succeq 0$$

and

$$\begin{bmatrix} aZ_1^{-,k} & \frac{b}{2}Z_2^{-,k} \\ \frac{b}{2}(Z_2^{-,k})^T & c\mathbf{Tr}(Z_1^{-,k}) \end{bmatrix} \succeq 0 \implies \begin{bmatrix} Z_1^{-,k} & Z_2^{-,k} \\ (Z_2^{-,k})^T & \mathbf{Tr}(Z_1^{-,k}) \end{bmatrix} \succeq 0$$

Therefore, with $a > 0$, $c > 0$, $b^2 - 4ac \geq 0$, any solution of problem (3.13) is also a solution of problem (3.5) when $\beta = 0$. □

Remark 3 (Rank condition for CQNNs)**.** *A solution of (3.13) is always a solution of (3.5) when $\beta = 0$, $a > 0$, $c > 0$, and $b^2 - 4ac \geq 0$. However, for it to also be a solution of the original non-convex training problem (3.4), the condition on the minimum number of filters of (3.6) must be met.*

Theorem 1. *The training of a CQNN with filter size f of the form (3.13) can be formulated as follows*

$$\begin{aligned} &\min_{\bar{Z}_1, \bar{Z}_2} \mathcal{L}(\hat{y}, y) \\ &\quad s.t.\ \hat{y}_i = \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} a\bar{Z}_1 & \frac{b}{2}\bar{Z}_2 \\ \frac{b}{2}(\bar{Z}_2)^T & c\bar{Z}_4 \end{bmatrix} \begin{bmatrix} x_i \\ 1 \end{bmatrix},\ i = 1, \ldots, N \\ &\qquad \bar{Z}_4 = \boldsymbol{Tr}(\bar{Z}_1) \\ &\qquad \bar{Z}_{1_{\{r,c\}}} = 0 \quad if \quad |r - c| \geq f \end{aligned} \tag{3.14}$$

Proof. For this proof, the sample index subscript i will be dropped for convenience. Observe that

$$x = \begin{bmatrix} x_{1:k-1}^T & \chi_k^T & x_{k+f:n}^T \end{bmatrix}^T$$

The input-output equation given in (3.13) is given as

$$\hat{y} = \sum_{k=1}^{K} \left(\begin{bmatrix} \chi_k \\ 1 \end{bmatrix}^T \begin{bmatrix} aZ_1^k & \frac{b}{2}Z_2^k \\ \frac{b}{2}(Z_2^k)^T & cZ_4^k \end{bmatrix} \begin{bmatrix} \chi_k \\ 1 \end{bmatrix} \right)$$

Which can be rewritten as

$$\hat{y} = \sum_{k=1}^{K} \left(\begin{bmatrix} x_{1:k-1} \\ \chi_k \\ x_{k+f:n} \\ 1 \end{bmatrix}^T \begin{bmatrix} O_{k-1,k-1} & O_{k-1,f} & O_{k-1,p} & O_{k-1,1} \\ O_{f,k-1} & aZ_1^k & O_{f,p} & \frac{b}{2}Z_2^k \\ O_{p,k-1} & O_{p,f} & O_{p,p} & O_{p,1} \\ O_{1,k-1} & \frac{b}{2}(Z_2^k)^T & O_{1,p} & cZ_4^k \end{bmatrix} \begin{bmatrix} x_{1:k-1} \\ \chi_k \\ x_{k+f:n}^T \\ 1 \end{bmatrix} \right)$$

$$= \sum_{k=1}^{K} \left(\begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} O_{k-1,k-1} & O_{k-1,f} & O_{k-1,p} & O_{k-1,1} \\ O_{f,k-1} & aZ_1^k & O_{f,p} & \frac{b}{2}Z_2^k \\ O_{p,k-1} & O_{p,f} & O_{p,p} & O_{p,1} \\ O_{1,k-1} & \frac{b}{2}(Z_2^k)^T & O_{1,p} & cZ_4^k \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} \right)$$

where

$$p = n - k - f + 1$$

Since the vectors now contain x instead of χ_k, they are no longer dependent on k and can be removed from the summation as follows

$$\hat{y} = \begin{bmatrix} x \\ 1 \end{bmatrix}^T \sum_{k=1}^{K} \left(\begin{bmatrix} O_{k-1,k-1} & O_{k-1,f} & O_{k-1,p} & O_{k-1,1} \\ O_{f,k-1} & aZ_1^k & O_{f,p} & \frac{b}{2}Z_2^k \\ O_{p,k-1} & O_{p,f} & O_{p,p} & O_{p,1} \\ O_{1,k-1} & \frac{b}{2}(Z_2^k)^T & O_{1,p} & cZ_4^k \end{bmatrix} \right) \begin{bmatrix} x \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} a\bar{Z}_1 & \frac{b}{2}\bar{Z}_2 \\ \frac{b}{2}(\bar{Z}_2)^T & c\bar{Z}_4 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}$$

where

$$\bar{Z}_1 = \sum_{k=1}^{K} \begin{bmatrix} O_{k-1,k-1} & O_{k-1,f} & O_{k-1,p} \\ O_{f,k-1} & Z_1^k & O_{f,p} \\ O_{p,k-1} & O_{p,f} & O_{p,p} \end{bmatrix}, \; \bar{Z}_2 = \sum_{k=1}^{K} \begin{bmatrix} O_{k-1,1} \\ Z_2^k \\ O_{p,1} \end{bmatrix}, \; \bar{Z}_4 = \sum_{k=1}^{K} Z_4^k$$

The non-zero entries of $\bar{Z}_1$ are contained in the main diagonal and the first $f-1$ diagonals above and below it, therefore, $\bar{Z}_{1_{\{r,c\}}} = 0$ if $|r-c| \geq f$. This is shown visually in Figure 3.4. Finally, the trace condition becomes

$$\mathbf{Tr}(\bar{Z}_1) = \mathbf{Tr}\left(\sum_{k=1}^{K}\begin{bmatrix} O_{k-1,k-1} & O_{k-1,f} & O_{k-1,p} \\ O_{f,k-1} & Z_1^k & O_{f,p} \\ O_{p,k-1} & O_{p,f} & O_{p,p} \end{bmatrix}\right) = \sum_{k=1}^{K}\mathbf{Tr}(Z_1^k) = \sum_{k=1}^{K} Z_4^k = \bar{Z}_4$$

□

Remark 4 (Number of weights)**.** *Recalling that Z_1 is symmetric and $Z_4 = \boldsymbol{Tr}(Z_1)$, the solution of (3.13) requires solving for $\frac{(f+3)(n-f+1)f}{2}$ weights. Comparatively, the solution of (3.14) requires $\frac{(2n-f+1)f}{2} + n$ weights, which is always less when $1 < f \leq n$.*

Remark 5 (Representational capacity)**.** *As shown by the input-output expression of problem (3.14), CQNNs represent a subset of QNNs. If $\mathcal{Q}$, $\mathcal{Q}_{qnn}$, and $\mathcal{Q}_{cqnn}$ are the sets of all quadratic forms, QNN quadratic forms, and CQNN quadratic forms, respectively, then $\mathcal{Q}_{cqnn} \subset \mathcal{Q}_{qnn} \subset \mathcal{Q}$. This is shown in Figure 3.3.*

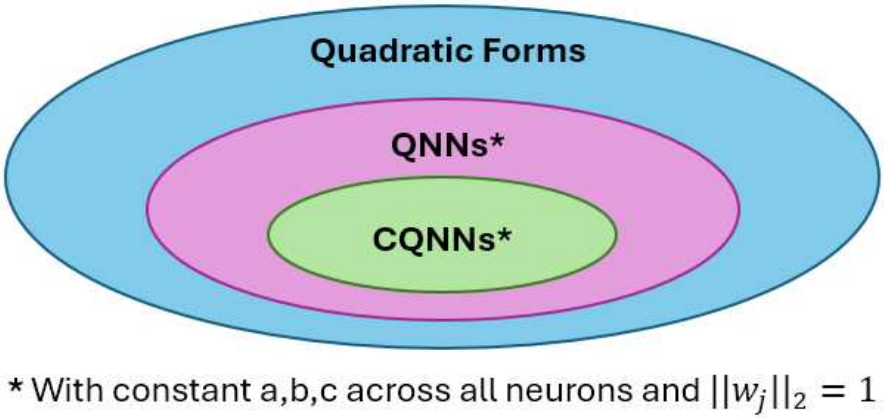


Figure 3.3: Sets of all quadratic forms, QNN quadratic forms, and CQNN quadratic forms

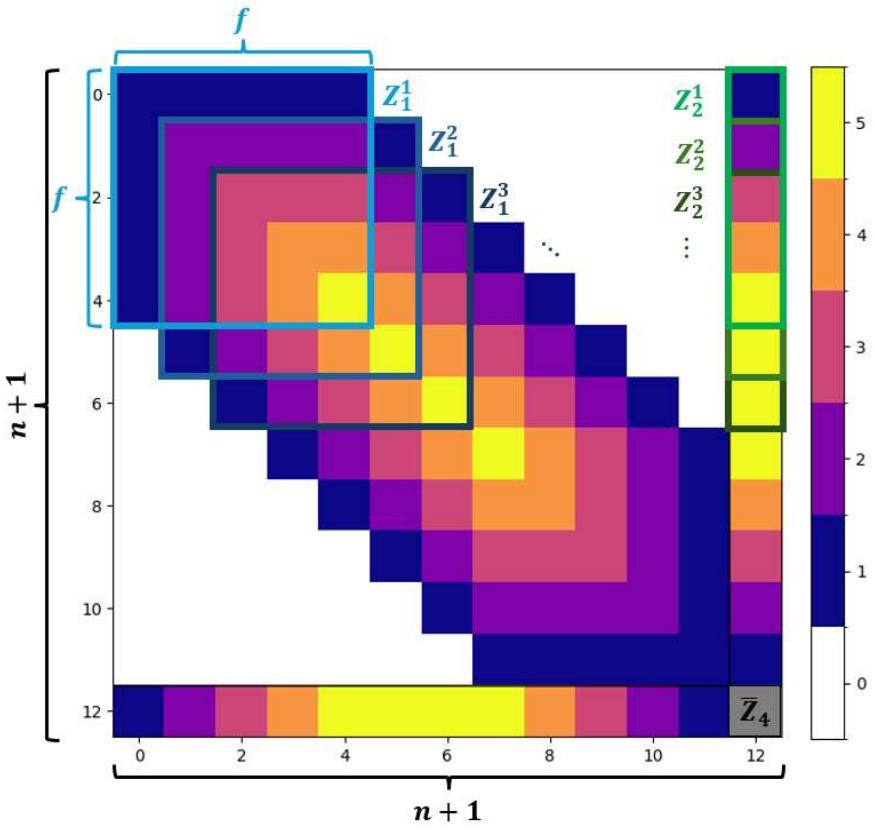


Figure 3.4: Visualization of $\bar{Z}_1$ and $\bar{Z}_2$, for $f = 5$ and $n = 12$. White squares represent zero entries, colored squares show the number of overlaps of different Z_1^k and Z_2^k to form the elements of $\bar{Z}_1$ and $\bar{Z}_2$

Corollary 1.1. *The sensitivity of the network's output as defined by (3.14) around a point* $x = x_0$ *is given by*

$$\left.\frac{\partial y}{\partial x}\right|_{x=x_0} = 2a\bar{Z}_1 x_0 + b\bar{Z}_2 \tag{3.15}$$

Proof. Taking the partial derivative of the network's output with respect to x gives

$$\frac{\partial y}{\partial x} = \frac{\partial}{\partial x}\left(\begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} a\bar{Z}_1 & \frac{b}{2}\bar{Z}_2 \\ \frac{b}{2}(\bar{Z}_2)^T & c\bar{Z}_4 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}\right) = \frac{\partial}{\partial x}\left(x^T a\bar{Z}_1 x + b\bar{Z}_2 x + c\bar{Z}_4\right) = 2a\bar{Z}_1 x + b\bar{Z}_2$$

□

Corollary 1.1 is used in the system identification example of Section 3.5.

3.3 Least Squares Training of CQNNs

Observing that the input-output expression of (3.14) is linear with respect to the elements of $\bar{Z}_1$ and $\bar{Z}_2$, the least squares training of a CQNN can be achieved in a similar manner to that of QNNs presented in Section 2.3. However, by taking advantage of the structure of $\bar{Z}_1$, the total number of weights that must be solved for can be reduced. Consider the matrix formed by xx^T using a vector $x \in \mathbb{R}^n$. As shown in Theorem 1, the matrix $\bar{Z}_1$ has zeros for the elements corresponding to $x_i x_j$ if $|i-j| \geq f$. Thus, the non-zero $x_i x_j$ combinations are all contained within the first f diagonals of the matrix xx^T starting from the main diagonal and going up.

Example 5 (Diagonals of xx^T)**.** *For the input vector $x = [x_1\ x_2\ x_3]^T$ and $f = 2$, xx^T is*

$$xx^T = \begin{bmatrix} x_1^2 & x_1x_2 & x_1x_3 \\ x_2x_1 & x_2^2 & x_2x_3 \\ x_3x_1 & x_3x_2 & x_3^2 \end{bmatrix}$$

and the non-zero elements are given by $diag_1 = [x_1^2 \quad x_2^2 \quad x_3^2]$ and $diag_2 = [x_1x_2 \quad x_2x_3]$.

Let $\mathbf{vecf}(xx^T, f)$ represent the row vector containing all the elements of the first f diagonals of matrix xx^T

$$\begin{aligned} \mathbf{vecf}(x_i x_i^T, f) &= \begin{bmatrix} diag_1 & diag_2 & \cdots & diag_f \end{bmatrix} \in \mathbb{R}^{1\times 0.5(2n-f+1)f} \\ diag_q &= \begin{bmatrix} x_{i_{\{1\}}}x_{i_{\{q\}}} & x_{i_{\{2\}}}x_{i_{\{q+1\}}} & \cdots & x_{i_{\{n-q+1\}}}x_{i_{\{n\}}} \end{bmatrix} \end{aligned} \tag{3.16}$$

where the number of columns of $\mathbf{vecf}(x_i x_i^T, f)$ is given by the length of the first f diagonals of $x_i x_i^T$, i.e,

$$(n) + (n-1) + (n-2) + \cdots + (n-f+1) = \frac{(2n-f+1)f}{2} \tag{3.17}$$

Consider a regressor matrix of the form

$$H = \begin{bmatrix} aH_1 + cH_2 & bX \end{bmatrix}$$
$$H_1 = \begin{bmatrix} \mathbf{vecf}(x_1 x_1^T, f) \\ \vdots \\ \mathbf{vecf}(x_N x_N^T, f) \end{bmatrix}, \quad H_2 = \begin{bmatrix} U_{N,n} & O_{N,0.5(2n-f+1)f-n} \end{bmatrix}, \quad X = \begin{bmatrix} x_1^T \\ \vdots \\ x_N^T \end{bmatrix} \tag{3.18}$$

Notice, this regressor matrix is similar to the one presented for QNN least squares training from Section 2.3. However, unlike the QNN regressor that uses all n diagonals of $x_i x_i^T$, the CQNN regressor only uses the first f diagonals. Using (3.18), problem (3.14) with loss function $\mathcal{L}(\hat{y}, y) = ||\hat{y} - y||^2$ becomes

$$\begin{aligned} &\min_{\theta} ||\hat{y} - y||^2 \\ &\text{s.t. } \hat{y} = H\theta \end{aligned} \tag{3.19}$$

with solution

$$\theta = (H^T H)^{-1} H^T y \tag{3.20}$$

The resulting weight vector θ is of the following form

$$\begin{aligned} \theta &= \begin{bmatrix} \theta_{1,1} & 2\theta_{1,2} & \cdots & 2\theta_{1,f} & \theta_2 \end{bmatrix}^T \in \mathbb{R}^{0.5(2n-f+1)f+n} \\ \theta_{1,i} &= \begin{bmatrix} \bar{Z}_{1_{\{1,i\}}} & \bar{Z}_{1_{\{2,i+1\}}} & \cdots & \bar{Z}_{1_{\{n-i+1,n\}}} \end{bmatrix}, \quad \theta_2 = \begin{bmatrix} \bar{Z}_{2_{\{1\}}} & \bar{Z}_{2_{\{2\}}} & \cdots & \bar{Z}_{2_{\{n\}}} \end{bmatrix} \end{aligned} \tag{3.21}$$

Regularization can also be introduced using regularized least squares to solve for the weight vector. The problem is given by

$$\begin{aligned} &\min_{\theta} ||\hat{y} - y||^2 + \beta\theta^T\theta \\ &\text{s.t. } \hat{y} = H\theta \end{aligned} \tag{3.22}$$

with solution

$$\theta = (H^T H + \beta I)^{-1} H^T y \tag{3.23}$$

where $\beta \geq 0$ is a regularization term.

Remark 6 (Regularized least squares)**.** *Since the regularization is being applied differently in the original convex formulation (2.8) and the least squares formulation (3.22), they will only share a solution when $\beta = 0$. However, using small values of β will allow for a good approximation of the optimal solution while ensuring $(H^T H + \beta I)$ is invertible.*

3.4 Multi-Channel CQNNs

Oftentimes, the input to a convolutional neural network will have multiple independent channels. These are referred to as multi-channel inputs. For example, a digital image is a multi-channel input, with a red, a green, and a blue channel, each representing the intensity of their respective color for each pixel. Another example is the data from an accelerometer, containing separate channels for the x, y, and z axes. Although not explicitly stated, the data patch definition used up until now considered a single-channel input. This section will extend the theory to explicitly handle the multi-channel case. Let x be an input vector consisting of C input channels $z_1, \ldots, z_C$, each of length n, that have been concatenated together.

$$x = \begin{bmatrix} z_1^T & \cdots & z_C^T \end{bmatrix}^T \in \mathbb{R}^{nC}, \quad z_c = \begin{bmatrix} z_{c_{\{1\}}} & \cdots & z_{c_{\{n\}}} \end{bmatrix}^T \in \mathbb{R}^n \tag{3.24}$$

When the filter passes over input x, it passes over each channel independently; that is to say, there cannot exist any data patches that share data from different channels. This is visualized in Figure 3.5. The result is a total of $K = C(n - f + 1)$ data patches, which are

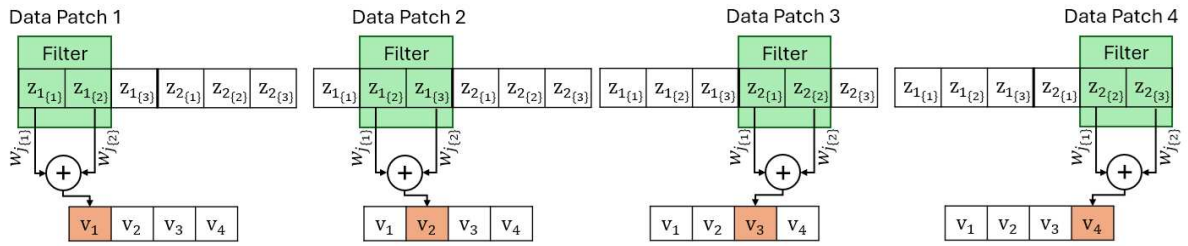


Figure 3.5: Visualization of convolutional layer feature extraction for a multi-channel input, $C = 2$ input channels, each of length $n = 3$, filter of length $f = 2$

defined as

$$\chi = \begin{bmatrix} \mathbb{X}_1^T & \mathbb{X}_2^T & \cdots & \mathbb{X}_c^T & \cdots & \mathbb{X}_C^T \end{bmatrix}^T$$
$$\mathbb{X}_c^T = \begin{bmatrix} \chi_{c,1}^T & \cdots & \chi_{c,n-f+1}^T \end{bmatrix}^T = \begin{bmatrix} \chi_{(c-1)(n-f+1)+1}^T & \cdots & \chi_{c(n-f+1)}^T \end{bmatrix}^T \tag{3.25}$$
$$\chi_{c,i} = \begin{bmatrix} z_{c_{\{i\}}} & z_{c_{\{i+1\}}} & \cdots & z_{c_{\{i+f-1\}}} \end{bmatrix}^T = z_{c_{\{i:i+f-1\}}}$$

where $\chi_{c,i} = \chi_{(c-1)(n-f+1)+i}$. The single subscript represents the global index (i.e., relative to the first patch of the first channel), whereas the double subscript represents the channel index (i.e., $\chi_{c,i}$ refers to the i^{th} data patch of the c^{th} channel).

Example 6. *Consider a 2-channel input with the first and second channels given by* $z_1 = [z_{1_{\{1\}}}\ z_{1_{\{2\}}}\ z_{1_{\{3\}}}]$ *and* $z_2 = [z_{2_{\{1\}}}\ z_{2_{\{2\}}}\ z_{2_{\{3\}}}]$*, respectively. The concatenated input is given by*

$$x = \begin{bmatrix} z_1^T & z_2^T \end{bmatrix}^T = \begin{bmatrix} z_{1_{\{1\}}} & z_{1_{\{2\}}} & z_{1_{\{3\}}} & z_{2_{\{1\}}} & z_{2_{\{2\}}} & z_{2_{\{3\}}} \end{bmatrix}^T$$

Passing a filter of length 2 *over the data produces the following data patches*

$$\chi_1 = \chi_{1,1} = \begin{bmatrix} z_{1_{\{1\}}} & z_{1_{\{2\}}} \end{bmatrix}^T, \quad \chi_2 = \chi_{1,2} = \begin{bmatrix} z_{1_{\{2\}}} & z_{1_{\{3\}}} \end{bmatrix}^T$$
$$\chi_3 = \chi_{2,1} = \begin{bmatrix} z_{2_{\{1\}}} & z_{2_{\{2\}}} \end{bmatrix}^T, \quad \chi_4 = \chi_{2,2} = \begin{bmatrix} z_{2_{\{2\}}} & z_{2_{\{3\}}} \end{bmatrix}^T$$

for a total of $K = C(n - f + 1) = 4$ *data patches with* $C = 2$*,* $f = 2$*, and* $n = 3$*.*

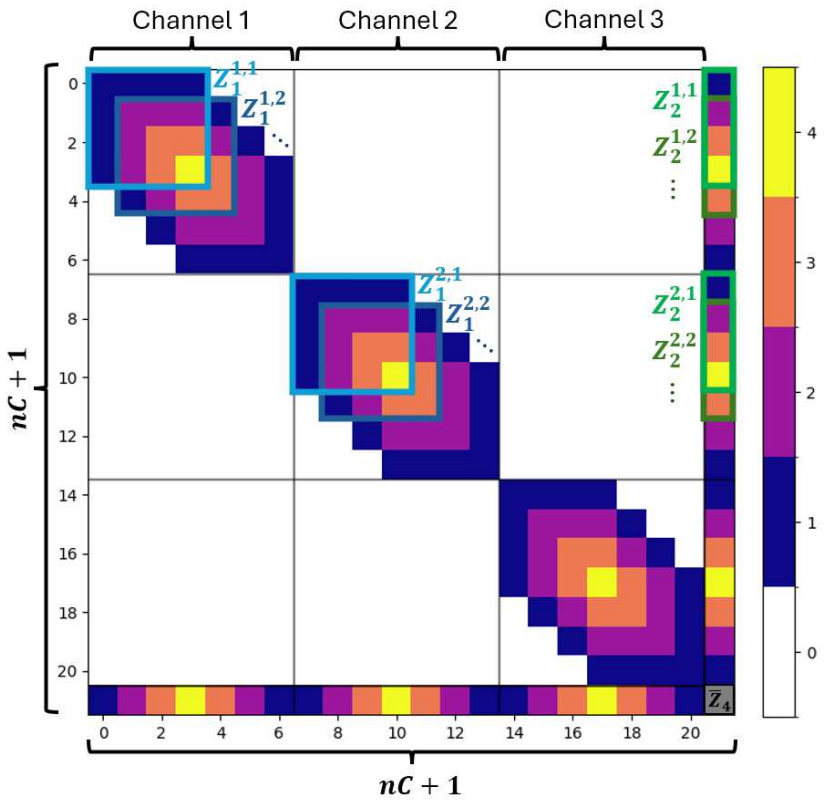


Figure 3.6: Visualization of Ξ_1 and Ξ_2, for $f = 4$, $C = 3$ and $n = 7$. White squares represent zero entries, colored squares show the number of overlaps of different $Z_1^{c,q}$ and $Z_2^{c,q}$ to form the elements of Ξ_1 and Ξ_2, respectively

Theorem 2. *The training of a CQNN with C input channels and with filter size f of the form (3.13) can be formulated as follows*

$$
\begin{aligned}
&\min_{\Xi_1,\Xi_2} \ \mathcal{L}(\hat{y}, y) \\
&s.t.\ \hat{y}_i = \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} a\Xi_1 & \frac{b}{2}\Xi_2 \\ \frac{b}{2}\Xi_2^T & c\Xi_4 \end{bmatrix} \begin{bmatrix} x_i \\ 1 \end{bmatrix}, \ i = 1, \dots, N \\
&\Xi_4 = \boldsymbol{Tr}(\Xi_1) \\
&\Xi_{1_{\{r,c\}}} = 0 \quad if \quad |r - c| \geq f, \\
&\Xi_{1_{\{r:r+n-1,c:c+n-1\}}} = O_{n,n} \quad if \quad r \neq c
\end{aligned}
\tag{3.26}
$$

Proof. For this proof, the sample index subscript i is dropped for convenience. Starting from the input-output expression given in (3.13), the summation is first expanded to sum over each of the channels, and then each of the data patches within each channel. For notational simplicity, the local notation for both the data patches and quadratic matrices is used (i.e., $\chi_{c,q}$ and $Z^{c,q}$ refer to the q^{th} data patch/matrix of the c^{th} channel).

$$\begin{aligned}\hat{y} &= \sum_{k=1}^{K} \left(\begin{bmatrix} \chi_k \\ 1 \end{bmatrix}^T \begin{bmatrix} aZ_1^k & \frac{b}{2}Z_2^k \\ \frac{b}{2}(Z_2^k)^T & cZ_4^k \end{bmatrix} \begin{bmatrix} \chi_k \\ 1 \end{bmatrix} \right) \\ &= \sum_{c=1}^{C} \sum_{q=1}^{n-f+1} \left(\begin{bmatrix} \chi_{c,q} \\ 1 \end{bmatrix}^T \begin{bmatrix} aZ_1^{c,q} & \frac{b}{2}Z_2^{c,q} \\ \frac{b}{2}(Z_2^{c,q})^T & cZ_4^{c,q} \end{bmatrix} \begin{bmatrix} \chi_{c,q} \\ 1 \end{bmatrix} \right)\end{aligned}$$

Next, by using the following definition

$$z_c = \begin{bmatrix} z_{c_{\{1:q-1\}}}^T & \chi_{c,q}^T & z_{c_{\{q+f:n\}}}^T \end{bmatrix}^T$$

the input-output expression can be re-written using z_c instead of $\chi_{c,q}$ using a similar padding technique as in Theorem 1, leading to

$$\hat{y} = \sum_{c=1}^{C} \left(\begin{bmatrix} z_c \\ 1 \end{bmatrix}^T \begin{bmatrix} a\bar{Z}_1^c & \frac{b}{2}\bar{Z}_2^c \\ \frac{b}{2}(\bar{Z}_2^c)^T & c\bar{Z}_4^c \end{bmatrix} \begin{bmatrix} z_c \\ 1 \end{bmatrix} \right) \tag{3.27}$$

with $p = n - q - f + 1$ and where

$$\bar{Z}_1^c = \sum_{q=1}^{n-f+1} \begin{bmatrix} O_{q-1,q-1} & O_{q-1,f} & O_{q-1,p} \\ O_{f,q-1} & Z_1^{c,q} & O_{f,p} \\ O_{p,q-1} & O_{p,f} & O_{p,p} \end{bmatrix}, \; \bar{Z}_2^c = \sum_{q=1}^{n-f+1} \begin{bmatrix} O_{q-1,1} \\ Z_2^{c,q} \\ O_{p,1} \end{bmatrix}, \; \bar{Z}_4^c = \sum_{q=1}^{n-f+1} Z_4^{c,q}$$

Finally, from (3.24) observe that

$$x = \begin{bmatrix} z_{1:c-1}^T & z_c^T & z_{c+1:C}^T \end{bmatrix}^T$$

Such that (3.27) can be rewritten as

$$\begin{aligned} \hat{y} &= \sum_{c=1}^{C} \left(\begin{bmatrix} z_{1:c-1} \\ z_c \\ z_{c+1:C} \\ 1 \end{bmatrix}^T \begin{bmatrix} O_{n(c-1),n(c-1)} & O_{n(c-1),n} & O_{n(c-1),n(C-c)} & O_{n(c-1),1} \\ O_{n,n(c-1)} & a\bar{Z}_1^c & O_{n,n(C-c)} & \frac{b}{2}\bar{Z}_2^c \\ O_{n(C-c),n(c-1)} & O_{n(C-c),n} & O_{n(C-c),n(C-c)} & O_{n(C-c),1} \\ O_{1,n(c-1)} & \frac{b}{2}(\bar{Z}_2^c)^T & O_{1,n(C-c)} & c\bar{Z}_4^c \end{bmatrix} \begin{bmatrix} z_{1:c-1} \\ z_c \\ z_{c+1:C} \\ 1 \end{bmatrix} \right) \\ &= \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} a\Xi_1 & \frac{b}{2}\Xi_2 \\ \frac{b}{2}(\Xi_2)^T & c\Xi_4 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} \end{aligned}$$

where

$$\Xi_1 = \begin{bmatrix} \bar{Z}_1^1 & O_{n,n} & \cdots & O_{n,n} \\ O_{n,n} & \bar{Z}_1^2 & \cdots & O_{n,n} \\ \vdots & \vdots & \ddots & \vdots \\ O_{n,n} & O_{n,n} & \cdots & \bar{Z}_1^C \end{bmatrix}, \quad \Xi_2 = \begin{bmatrix} \bar{Z}_2^1 \\ \bar{Z}_2^2 \\ \vdots \\ \bar{Z}_2^C \end{bmatrix}, \quad \Xi_4 = \sum_{c=1}^{C} \bar{Z}_4^c$$

Note the following two properties about the structure of the Ξ_1 matrix. First, $\Xi_{1_{\{r,c\}}} = 0$ if $|r - c| \geq f$. Second, Ξ_1 is block diagonal, composed of $n \times n$ submatrices. These two properties are shown visually in by example presented in Figure 3.6. Finally, the trace condition still holds, i.e.,

$$\mathbf{Tr}(\Xi_1) = \sum_{c=1}^{C} \mathbf{Tr}(\bar{Z}_1^c) = \sum_{c=1}^{C} \sum_{q=1}^{n-f+1} \mathbf{Tr}(Z_1^{c,q}) = \sum_{c=1}^{C} \sum_{q=1}^{n-f+1} Z_4^{c,q} = \sum_{c=1}^{C} \bar{Z}_4^c = \Xi_4$$

□

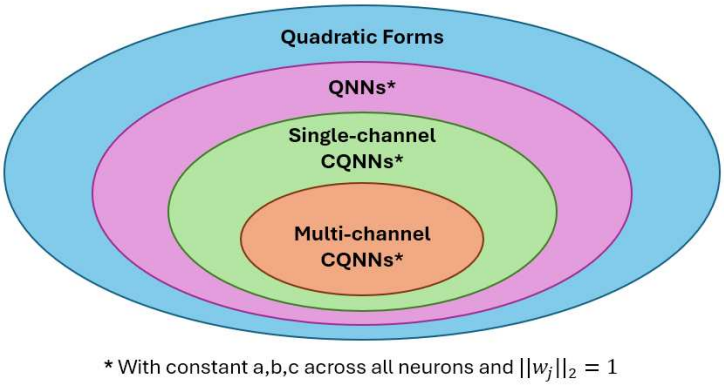


Figure 3.7: Sets of all quadratic forms, QNN quadratic forms, and single-channel/multi-channel CQNN quadratic forms

Compared to Theorem 1 for single-channel inputs, the multi-channel formulation uses a second iteration of zero padding to reach the full input vector x. The decision to the single-channel or multi-channel formulation is a choice of the user, as both formulations accept the same concatenated input vector x. However, when using the multi-channel formulation, the total number of weights to be computed is lower compared to the single-channel formulation. A comparison between the single-channel and multi-channel CQNNs is shown in the sensor fusion example of Section 3.5.

Remark 7 (Multi-channel weights)**.** *Recalling that Ξ_1 is symmetric and $\Xi_4 = \boldsymbol{Tr}(\Xi_1)$, the solution to (3.26) requires solving for $\frac{f(2n-f+1)C}{2} + nC$ weights for an input vector $x \in \mathbb{R}^{nC}$. Comparatively, using the same input in the single-channel formulation of (3.14) requires solving for $\frac{f(2nC-f+1)}{2} + nC$ weights. Thus, using the multi-channel form reduces the total required weights to be computed by $\frac{f(f-1)(C-1)}{2}$.*

Remark 8 (Representational capacity)**.** *As shown by the input-output expression and constraints of problem (3.26), multi-channel CQNNs represent a subset of single-channel CQNNs. If $\mathcal{Q}$, $\mathcal{Q}_{qnn}$, $\mathcal{Q}_{cqnn}$, and $\mathcal{Q}_{mcqnn}$ are the sets of all quadratic forms, QNN quadratic forms, single-channel CQNN quadratic forms, and multi-channel CQNN quadratic forms, respectively, then $\mathcal{Q}_{mcqnn} \subset \mathcal{Q}_{cqnn} \subset \mathcal{Q}_{qnn} \subset \mathcal{Q}$. This is shown in Figure 3.7.*

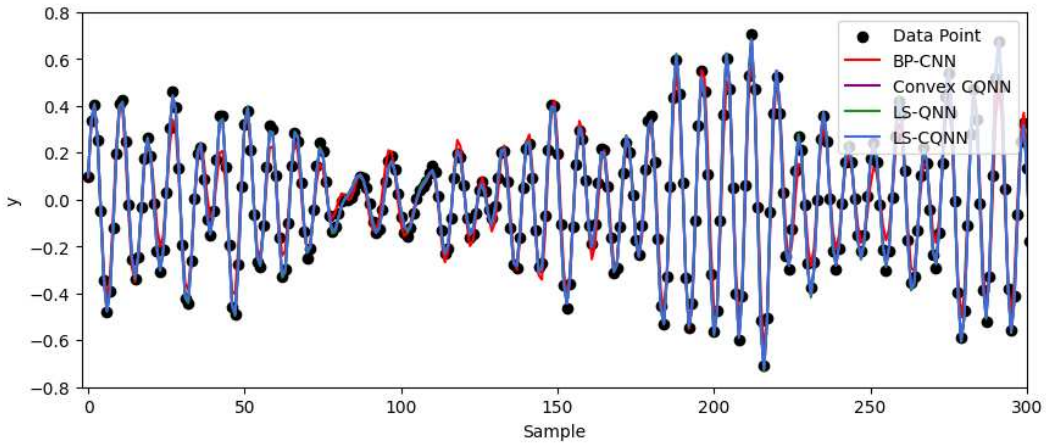


Figure 3.8: System identification comparison between different networks over first 300 samples from test set

3.5 Examples

For all examples, the PyTorch library in Python was used for the backpropagation-trained networks (BP-CNN). All BP-CNNs have two layers and use the ReLU activation function in their hidden layer. Additionally, the BP-CNNs were trained with the stochastic gradient descent algorithm using a learning rate of 0.01 and a momentum factor of 0.9. All quadratic networks use $a = 0.0937$, $b = 0.5$, $c = 0.4688$ for their activation function parameters following the optimal mean squared error ReLU approximation over the interval $[-5, 5]$, which is presented in Section 5.1.

3.5.1 Example 1: System Identification of Flexible Robot Arm

System identification of a flexible robot arm was performed, with the data being available from DaISy (Database for the Identification of System) [47]. Each input sample X_i and output sample Y_i was constructed as follows:

$$X_i = \begin{bmatrix} u_{i-d} & \cdots & u_{i-1} & y_{i-d} & \cdots & y_{i-1} \end{bmatrix} \tag{3.28a}$$

$$Y_i = \begin{bmatrix} y_i \end{bmatrix} \tag{3.28b}$$

where u_i is the arm reaction torque, y_i is the arm acceleration, and d is a delay term that dictates how many past samples are used to predict the next output. For this example, $d = 6$ was selected, and the 1024 samples were divided using a 40/60 train/test split. All convolutional networks used a filter size of 3, and the BP-CNN used 128 filters, the ReLU activation function, and was trained over 10k epochs. Along with the BP-CNN, the compared methods were the CQNN convex training method (Convex CQNN) proposed in [12], the least squares QNN (LS-QNN) proposed in [21], and the least squares CQNN (LS-CQNN) from this thesis. Both the LS-QNN and LS-CQNN used $\beta = 10^{-10}$ as their regularization coefficient.

The results of the models being used in an auto-regressive fashion over the test data are plotted in Figure 3.8 and the absolute error of each network is shown in Figure 3.9. The training mean squared error (MSE), testing MSE, and training time are shown in Table 3.1. The LS-CQNN outperformed the BP-CNN in both testing MSE and training time. Additionally, compared to the LS-QNN, the LS-CQNN had a larger training MSE but a lower testing MSE. This suggests that the added constraints of LS-CQNNs (see Remark 5 of Section 3.2) act as a form of regularization which aids in the generalization of the network. Finally, compared to a BP-trained network, the LS-CQNN method provides an analytic quadratic model of the system. Using this quadratic model and equation (3.15), the maximum sensitivity of the output to each input is found as

$$\frac{\partial y_i}{\partial u_{i-d}} = \begin{bmatrix} -0.00507 & -0.109 & 0.309 & -0.414 & 0.329 & -0.119 \end{bmatrix}^T$$
$$\frac{\partial y_i}{\partial y_{i-d}} = \begin{bmatrix} -0.257 & 0.906 & -1.244 & 1.03 & -0.900 & 0.675 \end{bmatrix}^T$$

This shows that the system is more sensitive to previous outputs compared to inputs, and that the output is affected mostly by the input after a few delays, suggesting a possible lag between the input and the output of the system.

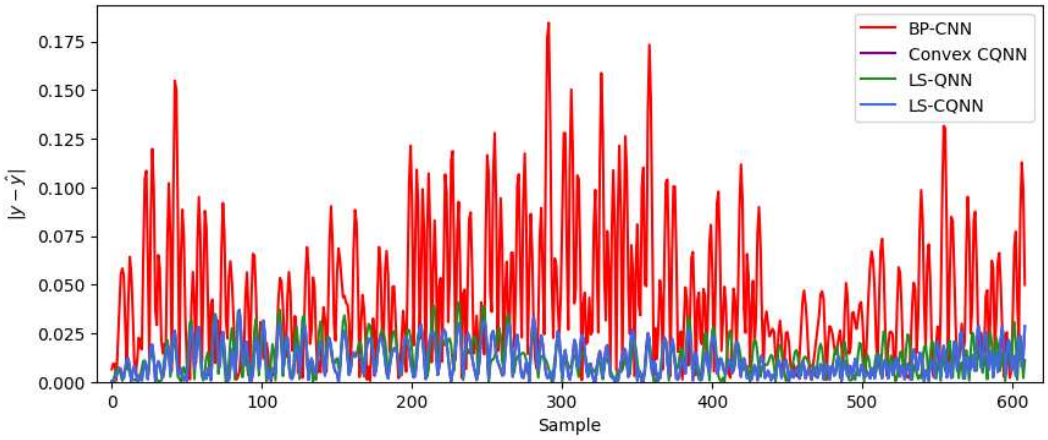


Figure 3.9: System identification comparison of absolute error between different networks over testing data

Table 3.1: System identification auto-regressive training/testing MSE and training time for various neural network solutions

Network	Training MSE	Testing MSE	Training Time [s]
LS-QNN [21]	5.78×10^{-5}	2.40×10^{-4}	0.040
LS-CQNN	6.88×10^{-5}	1.90×10^{-4}	0.026
Convex CQNN [12]	6.89×10^{-5}	1.91×10^{-4}	16.981
BP-CNN	2.57×10^{-3}	3.22×10^{-3}	54.128

3.5.2 Example 2: GPS Signal Emulation

Typical outdoor drone navigation uses both an IMU (inertial measurement unit) and GPS (global positioning system) data for sensor fusion to produce accurate position and attitude estimations. GPS outages present an issue for this type of sensor fusion, as when GPS measurements are not present, the estimation errors grow rapidly. Recent work has proposed the training of neural networks to mimic the GPS signal during periods of outages [48, 49, 50]. For this example, both a single-channel CQNN and a multi-channel CQNN will be used on synthetic drone flight data created from the NaveGo toolbox for MATLAB [51, 52]. The trajectory is shown in Figure 3.10. Following the model proposed in [48], the input and

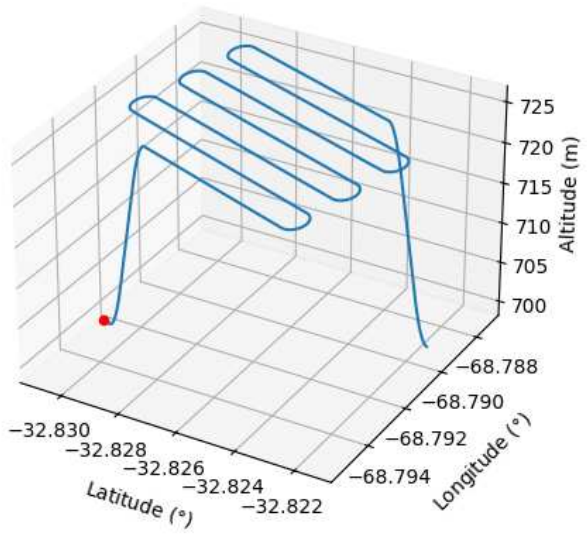


Figure 3.10: Synthetic drone trajectory with starting position indicated in red

output vectors have the following form

$$X_i = \begin{bmatrix} a_x^T & a_y^T & a_z^T & \omega_x^T & \omega_y^T & \omega_z^T & \theta_x^T & \theta_y^T & \theta_z^T \end{bmatrix}^T \tag{3.29a}$$

$$Y_i = \begin{bmatrix} \Delta P_{lat} & \Delta P_{lon} \end{bmatrix}^T \tag{3.29b}$$

where $a_x, a_y, a_z \in \mathbb{R}^r$, $\omega_x, \omega_y, \omega_z \in \mathbb{R}^r$, and $\theta_x, \theta_y, \theta_z \in \mathbb{R}^r$ are the IMU acceleration, IMU angular rate, and INS (inertial navigation system) attitude in the x, y, z axes of the body frame of the the drone and r is the sampling rate ratio between the IMU and the GPS. Furthermore, ΔP_{lat} and ΔP_{lon} are the change in latitude and longitude of the drone in the navigation frame between subsequent GPS measurements. The GPS operates at 5 Hz, whereas the IMU and INS operate at 200 Hz, leading to $r = 40$ for this example. The data contains a total of 2186 samples, which are divided using a 50/50 train/test split. To compare with the CQNNs, a backpropagation-trained CNN (BP-CNN) was trained on the data over 2000 epochs, using 81 filters, and a weight decay of 0.01. Both lhe least squares single-channel CQNN (LS-CQNN) and multi-channel CQNN (LS-CQNN Multi) were trained with $\beta = 100$. All networks used a filter size of 7.

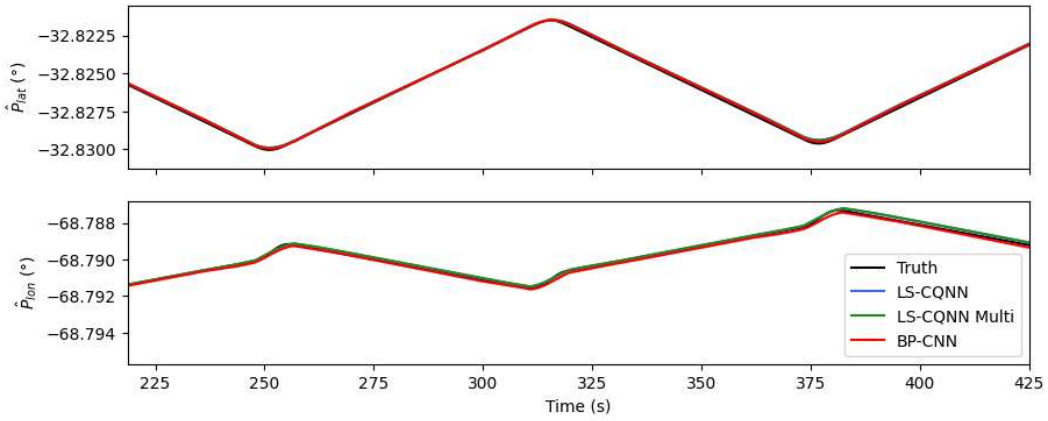


Figure 3.11: Predicted GPS latitude and longitude using BP-CNN and LS-CQNN

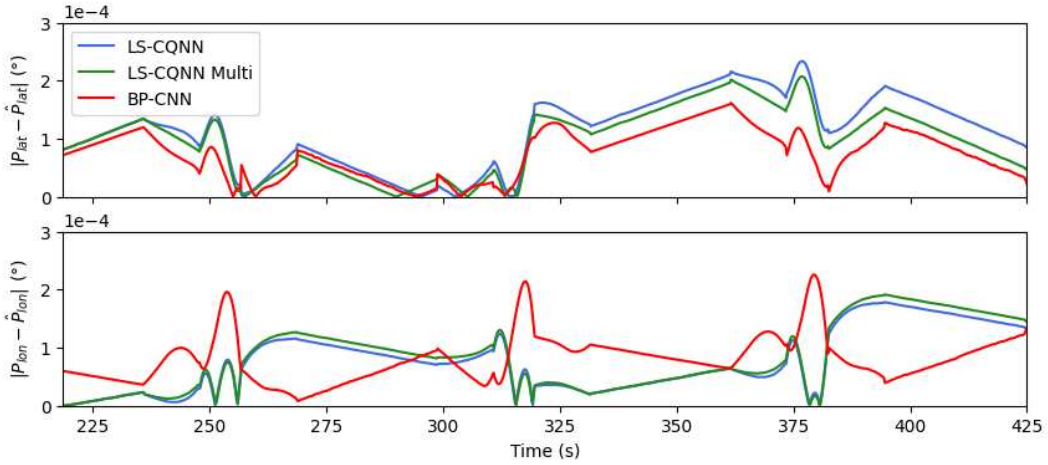


Figure 3.12: Predicted GPS latitude and longitude absolute error using BP-CNN and LS-CQNN

The results for latitude and longitude predictions for the test set are shown in Figure 3.11, with the absolute error of the predictions shown in Figure 3.12. Both LS-CQNNs and the BP-CNN achieved similar performances in terms of predictive error. However, the LS-CQNNs both had a significant reduction in training time, with the BP-CNN taking 14.103 seconds and the single-channel/multi-channel CQNNs taking 2.154 seconds and 1.924 seconds, respectively. Moreover, this shows the advantage of the multi-channel formulation with a shorter training time due to reduced number of weights that must be computed (see Remark 7, Section 3.4). In addition to a faster training time, the LS-CQNNs would also permit a sensitivity analysis of the model, unlike the BP-CNN.

Chapter 4

Least Squares Training of Quadratic Physics-Informed Neural Networks

This chapter presents the methodology to formulate the training of a two-layer quadratic physics-informed neural network (QPINN) as a least squares problem. The chapter is structured as follows. Section 4.1 provides an overview of PINNs and their usage for solving boundary value problems (BVPs). Section 4.2 presents the least squares formulation of QPINN training. Section 4.3 gives the required background on partial differential equation (PDE) symmetries and the symmetry loss term for PINNs. Section 4.4 provides the required conditions and methodology for integrating PDE symmetries into the least squares training of QPINNs. Finally, Section 4.5 applies QPINNs to various BVPs.

4.1 Physics-Informed Neural Networks

Physics-informed neural networks (PINNs) are a type of neural network that are used to approximate the solution of boundary value problems (BVPs) (i.e., a set of differential equations subject to a set of boundary conditions). Unlike conventional neural networks, PINNs do not require data for training. Instead, they sample the input space to enforce satisfaction of the underlying differential equation and boundary conditions to train the network.

4.1.1 Partial Differential Equations

We consider boundary value problems of the following from

$$\begin{aligned} \Delta = \mathcal{D}^{(n)}[u] + f(t,x) = 0, \quad & t \in [0,T], x \in \Omega \\ u(0,x) = g(x), \quad & x \in \Omega \\ u(t,x) = h(t,x), \quad & t \in [0,T], x \in \partial\Omega \end{aligned} \tag{4.1}$$

where $t \in [0,T]$ is the temporal independent variable, $x \in \Omega \subset \mathbb{R}^p$ are the spatial independent variables, and $u \in \mathbb{R}^q$ are the dependent variables. Additionally, $g(x)$ is the initial condition function, and $h(t,x)$ is the boundary condition function applied to the boundary region $\partial\Omega$. Furthermore, Δ is a set of M partial differential equations (PDEs), composed of a linear differential operator $\mathcal{D}^{(n)}[\cdot]$ with maximum order n, and $f(t,x)$, a function which does not depend on the dependent variables or their derivatives. Note, that in general, since there are M differential equations and q dependent variables, the differential operator $\mathcal{D}^{(n)}[\cdot]$ takes the form of a matrix of differential operators, this is illustrated in Example 7.

Example 7 (Differential operator $\mathcal{D}^{(n)}$)**.** *Let $x = [x_1\ x_2]^T$ and $u = [u_1\ u_2]^T$, consider a set of two partial differential equations given by*

$$\partial_{x_1} u_1 + \partial_{x_2} u_2 + x_1 = 0$$

$$\partial_{x_2 x_2} u_1 + \partial_{x_1 x_1} u_2 = 0$$

which can be written in the form of Δ from (4.1) as

$$\Delta = \mathcal{D}^{(n)}[u] + f(t,x) = \begin{bmatrix} \partial_{x_1} & \partial_{x_2} \\ \partial_{x_2 x_2} & \partial_{x_1 x_1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} x_1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

where it can be seen that $\mathcal{D}^{(n)}[\cdot]$ is a matrix of differential operators.

Example 8 (Heat equation BVP)**.** *The diffusion of heat in a material is governed by a PDE called the heat equation. In the 1-dimensional case, the equation is given as*

$$\partial_t u = \nu \partial_{xx} u$$

where t is the time, x is the position along the material, $u(t, x)$ is the temperature, and $\nu > 0$ is the thermal diffusivity coefficient. Consider the region $x \in [0, 2\pi]$ and $t \in [0, 20]$, with initial temperature distribution $g(x) = \sin(x) + 1$ and fixed boundary temperature $h(t, x) = 1$. Solving for $u(t, x)$ requires solving the following BVP problem

$$\begin{aligned} \Delta = \partial_t u - \nu \partial_{xx} u = 0, \quad & t \in [0, 20], x \in [0, 2\pi] \\ u(0, x) = \sin(x) + 1, \quad & x \in [0, 2\pi] \\ u(t, x) = 1, \quad & t \in [0, 20], x \in \{0, 2\pi\} \end{aligned}$$

4.1.2 PINN Loss Function

In contrast to conventional neural networks, which use a loss that penalizes differences between predicted outputs and labels, PINNs use a loss function that penalizes deviation from the underlying PDE and the initial/boundary conditions [34, 35]. The PINN loss function takes the form

$$\mathcal{L} = \gamma_{pde}\mathcal{L}_{pde} + \gamma_{ic}\mathcal{L}_{ic} + \gamma_{bc}\mathcal{L}_{bc} + \beta\mathcal{L}_{reg} \tag{4.2}$$

where $\mathcal{L}_{pde}$ is the PDE deviation loss, $\mathcal{L}_{ic}$ is the initial condition loss, $\mathcal{L}_{bc}$ is the boundary condition loss, and $\gamma_{pde} \geq 0, \gamma_{ic} \geq 0, \gamma_{bc} \geq 0$ are weighing terms for each of the respective losses. Furthermore, $\mathcal{L}_{reg}$ is a regularization loss that is modulated using $\beta > 0$. With the exception of $\mathcal{L}_{reg}$, each loss term is applied to a set of points (t_i, x_i) which are sampled from the input space (t, x). An example of this sampling is shown in Figure 4.1. For a predicted

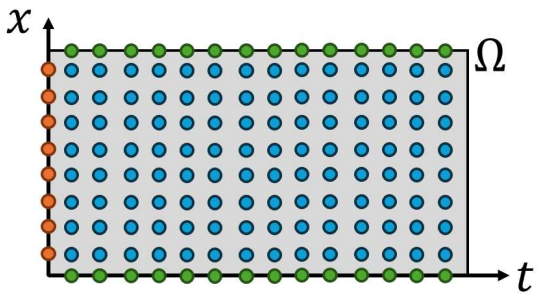


Figure 4.1: Example of loss function sampling for a system with a scalar spatial variable x, $\mathcal{L}_{pde}$ samples over entire region Ω shown in blue, $\mathcal{L}_{ic}$ samples at $t = 0$ shown in orange, and $\mathcal{L}_{bc}$ samples on the spatial boundaries shown in green

solution $\hat{u}$, the PDE loss function is given by

$$\mathcal{L}_{pde} = \frac{1}{N_p} \sum_{i=1}^{N_p} \|\mathcal{D}^{(n)}[\hat{u}(t_i, x_i)] + f(t_i, x_i)\|_2^2 \tag{4.3}$$

where $\{(t_i, x_i)\}_{i=1}^{N_p}$ are N_p points sampled for $t \in [0, T]$ and $x \in \Omega$. The initial condition loss is given by

$$\mathcal{L}_{ic} = \frac{1}{N_i} \sum_{i=1}^{N_i} \|\hat{u}(0, x_i) - g(x_i)\|_2^2 \tag{4.4}$$

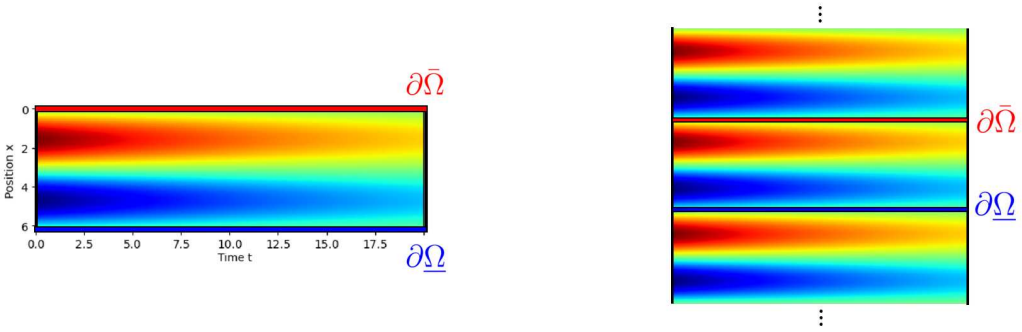
where $\{x_i\}_{i=1}^{N_i}$ are N_i points sampled for $x \in \Omega$. Finally, depending on whether the boundary conditions are periodic, Dirichlet, or Neumann, the boundary condition loss is

$$\text{Periodic:} \quad \mathcal{L}_{bc} = \frac{1}{N_b} \sum_{i=1}^{N_b} \|\hat{u}(t_i, \overline{x}_i) - \hat{u}(t_i, \underline{x}_i)\|_2^2 \tag{4.5a}$$

$$\text{Dirichlet:} \quad \mathcal{L}_{bc} = \frac{1}{N_b} \sum_{i=1}^{N_b} \|\hat{u}(t_i, x_i) - h(t_i, x_i)\|_2^2 \tag{4.5b}$$

$$\text{Neumann:} \quad \mathcal{L}_{bc} = \frac{1}{N_b} \sum_{i=1}^{N_b} \|\partial_n \hat{u}(t_i, x_i) - h(t_i, x_i)\|_2^2 \tag{4.5c}$$

where for Dirichlet and Neumann conditions $\{(t_i, x_i)\}_{i=1}^{N_b}$ are N_b points sampled for $t \in [0, T]$ and $x \in \partial\Omega$, and for periodic conditions $\{(t_i, \underline{x}_i, \overline{x}_i)\}_{i=1}^{N_b}$ are N_b points sampled for $t \in [0, T]$, $\underline{x} \in \partial\underline{\Omega}$ and $\overline{x} \in \partial\overline{\Omega}$, where $\partial\underline{\Omega}$ and $\partial\overline{\Omega}$ are regions of the border of Ω which are connected



(a) Heat equation solution over $x \in [0, 2\pi]$ and $t \in [0, 20]$, spatial bounds indicated by red and blue boundaries

(b) Periodic tiling of solution to form boundary conditions

Figure 4.2: Periodic tiling boundary conditions for the 1-dimensional heat equation

via periodic tiling. An example of periodic tiling is shown in Figure 4.2. Furthermore, in equation (4.5c), ∂_n is the normal derivative with respect to the boundary defined as

$$\begin{aligned} \partial_n \hat{u} &= \begin{bmatrix} \partial_n \hat{u}_1 & \cdots & \partial_n \hat{u}_q \end{bmatrix} \\ \partial_n \hat{u}_\alpha = \nabla_x \hat{u}_\alpha \cdot n &= \begin{bmatrix} \partial_{x_1} \hat{u}_\alpha & \cdots & \partial_{x_p} \hat{u}_\alpha \end{bmatrix}^T \cdot n \end{aligned} \tag{4.6}$$

where n is the external unit normal at the boundary point where the the derivative is computed.

Example 9 (Boundary condition types)**.** *Consider the BVP of the 1-dimensional heat equation with predicted solution $\hat{u}(t, x)$ over the spatial interval $x \in [0, 2\pi]$ with initial condition $g(x) = \sin(x) + 1$. The following are examples of each of the boundary condition types, along with how they would be implemented into their respective loss terms. Periodic boundary conditions connecting both ends of the material at $\underline{x} = 0$ and $\overline{x} = 2\pi$ are given by*

$$\hat{u}(t, 0) = \hat{u}(t, 2\pi) \tag{4.7}$$

where using (4.7) in (4.5a) yields

$$\|\hat{u}(t_i, \bar{x}_i) - \hat{u}(t_i, \underline{x}_i)\|_2^2 = \|\hat{u}(t_i, 0) - \hat{u}(t_i, 2\pi)\|_2^2$$

This periodic tiling is shown in Figure 4.2. Dirichlet boundary conditions for a fixed temperature at the boundary (i.e., $h(t_i, 0) = h(t_i, 2\pi) = 1$) are given by

$$\hat{u}(t, 0) = 1, \quad \hat{u}(t, 2\pi) = 1 \tag{4.8}$$

where using (4.8) in (4.5b) yields

$$\|\hat{u}(t_i, x_i) - h(t_i, x_i)\|_2^2 = \|\hat{u}(t_i, 0) - 1\|_2^2$$
$$\|\hat{u}(t_i, x_i) - h(t_i, x_i)\|_2^2 = \|\hat{u}(t_i, 2\pi) - 1\|_2^2$$

Neumann boundary conditions for no temperature flux at the boundary (i.e., $h(t_i, x_i) = 0$) are given by

$$\partial_n u(t, 0) = -\partial_x u(t, 0) = 0, \quad \partial_n u(t, 2\pi) = \partial_x u(t, 2\pi) = 0 \tag{4.9}$$

where using (4.9) in (4.5c) yields

$$\|\partial_n \hat{u}(t_i, x_i) - h(t_i, x_i)\|_2^2 = \|\partial_x \hat{u}(t_i, 0)\|_2^2$$
$$\|\partial_n \hat{u}(t_i, x_i) - h(t_i, x_i)\|_2^2 = \|\partial_x \hat{u}(t_i, 2\pi)\|_2^2$$

4.2 Least Squares Training of QPINNs

From the least squares formulation for training a QNN presented in problem (2.11), the output of a two-layer QNN being trained using N data points is given as

$$\hat{y} = H\theta \tag{4.10}$$

where θ is a weight vector and H is a regressor matrix formed using the training points $\{x_i\}_{i=1}^N$ given by

$$H = H(x) = \begin{bmatrix} \zeta(x_1) & \cdots & \zeta(x_N) \end{bmatrix}^T \tag{4.11}$$

with $\zeta(x_i)^T$ being the i^{th} row of the regressor matrix H. The input-output expression for a single sample is therefore given as

$$\hat{y}_i = \zeta^T(x_i)\theta \tag{4.12}$$

For the approximation of the solution to a boundary value problem, the input-output expression (4.12) is instead written as

$$\hat{u}_i = \zeta^T(t_i, x_i)\theta \tag{4.13}$$

where the output variable is replaced by $\hat{u}$ and the input is split into its temporal component $t_i \in \mathbb{R}$ and spatial components $x_i \in \mathbb{R}^p$.

4.2.1 Multiple Dependent Variables

The input-output expression of (4.13) is for a single output. For BVPs with multiple independent variables $u_1, \ldots, u_q$, let each be approximated using a QNN of the form

$$\hat{u}_\alpha = \zeta_\alpha^T(t, x)\theta_\alpha \tag{4.14}$$

where θ_α and ζ_α are the weight vector and the regressor vector associated with the output of the α^{th} QNN. The vector of all outputs $\hat{u} = [\hat{u}_1 \ \cdots \ \hat{u}_q]^T$ can be written as

$$\hat{u} = \zeta^T(t, x)\theta \tag{4.15}$$

where θ is the concatenated weight vector given as

$$\theta = \begin{bmatrix} \theta_1^T & \cdots & \theta_q^T \end{bmatrix}^T \tag{4.16}$$

and ζ^T is a regressor matrix of the form

$$\zeta^T(t,x) = \begin{bmatrix} \zeta_1^T(t,x) & O_{1,r_2} & \cdots & O_{1,r_q} \\ O_{1,r_1} & \zeta_2^T(t,x) & \cdots & O_{1,r_q} \\ \vdots & \vdots & \ddots & \vdots \\ O_{1,r_1} & O_{1,r_2} & \cdots & \zeta_q^T(t,x) \end{bmatrix} \tag{4.17}$$

where r_α is the length of the regressor vector ζ_α. Therefore, using (4.15) QPINNs can be used for BVPs with multiple dependent variables, with all networks trained simultaneously.

Example 10 (Continuity PDE)**.** *The continuity PDE for steady-state flow is given by*

$$\Delta = \partial_{x_1} u_1 + \partial_{x_2} u_2 = 0 \tag{4.18}$$

where $x = [x_1\ x_2]^T$ are positions, and $u = [u_1\ u_2]^T$ are the velocities. Using (4.14), the QNN approximated solutions of u_1 and u_2 are given by

$$\hat{u}_1 = \zeta_1^T(x)\theta_1, \quad \hat{u}_2 = \zeta_2^T(x)\theta_2$$

such that $\hat{u}$ is given by

$$\hat{u} = \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \end{bmatrix} = \begin{bmatrix} \zeta_1^T(x)\theta_1 \\ \zeta_2^T(x)\theta_2 \end{bmatrix} = \begin{bmatrix} \zeta_1^T(x) & O_{1,r_2} \\ O_{1,r_1} & \zeta_2^T(x) \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \zeta^T(x)\theta$$

where r_1 and r_2 are the lengths of regressor vectors $\zeta_1(x)$ and $\zeta_2(x)$, respectively.

4.2.2 QPINN Loss

This subsection provides the methodology for analytic training of QPINNs with the loss function defined in equation (4.2).

4.2.2.1 PDE Loss Term

Consider the BVP defined in (4.1) that is composed of M differential equations. The PDE loss term from (4.3) is given as

$$\begin{aligned}\mathcal{L}_{pde} &= \frac{1}{N_p}\sum_{i=1}^{N_p}\|\mathcal{D}^{(n)}[\hat{u}(t_i,x_i)] + f(t_i,x_i)\|_2^2 \\ &= \frac{1}{N_p}\sum_{i=1}^{N_p}\sum_{m=1}^{M}\left(\mathcal{D}_m^{(n)}[\hat{u}(t_i,x_i)] + f_m(t_i,x_i)\right)^2\end{aligned} \tag{4.19}$$

where $\mathcal{D}_m^{(n)}[\cdot]$ and $f_m(\cdot)$ are the components of $\mathcal{D}^{(n)}[\cdot]$ and $f(\cdot)$ associated with the m^{th} PDE. Substituting the QNN input-output expression (4.15) into (4.19) yields

$$\mathcal{L}_{pde} = \frac{1}{N_p}\sum_{i=1}^{N_p}\sum_{m=1}^{M}\left(\mathcal{D}_m^{(n)}[\zeta^T(t_i,x_i)]\theta + f_m(t_i,x_i)\right)^2 \tag{4.20}$$

where θ is moved outside the differential operator as it is neither a function of t nor x. The PDE loss term can thus be written as

$$\mathcal{L}_{pde} = \frac{1}{N_p}\sum_{m=1}^{M}\left(Y_m^p - R_m^p\theta\right)^T\left(Y_m^p - R_m^p\theta\right) \tag{4.21}$$

where

$$R_m^p = \begin{bmatrix}\mathcal{D}_m^{(n)}[\zeta^T(t_1,x_1)] \\ \vdots \\ \mathcal{D}_m^{(n)}[\zeta^T(t_{N_p},x_{N_p})]\end{bmatrix}, \quad Y_m^p = \begin{bmatrix}-f_m(t_1,x_1) \\ \vdots \\ -f_m(t_{N_p},x_{N_p})\end{bmatrix} \tag{4.22}$$

4.2.2.2 Initial Condition Loss Term

Using a similar methodology as presented for the PDE loss, the initial condition loss term from (4.4) can be rewritten as

$$\mathcal{L}_{ic} = \frac{1}{N_i} \left(Y^i - R^i\theta\right)^T \left(Y^i - R^i\theta\right) \tag{4.23}$$

where

$$R^i = \begin{bmatrix} \zeta^T(0, x_1) \\ \vdots \\ \zeta^T(0, x_{N_i}) \end{bmatrix}, \quad Y^i = \begin{bmatrix} g(x_1) \\ \vdots \\ g(x_{N_i}) \end{bmatrix} \tag{4.24}$$

4.2.2.3 Boundary Condition Loss Term

Based on the type of boundary condition, the boundary condition loss term from (4.5a), (4.5b), or (4.5c) can be rewritten as

$$\mathcal{L}_{bc} = \frac{1}{N_b} \left(Y^b - R^b\theta\right)^T \left(Y^b - R^b\theta\right) \tag{4.25}$$

where, depending on the type of boundary condition, the following R^b and Y^b are used,

$$\text{Dirichlet:} \quad R^b = \begin{bmatrix} \zeta^T(t_1, x_1) \\ \vdots \\ \zeta^T(t_{N_b}, x_{N_b}) \end{bmatrix}, \quad Y^b = \begin{bmatrix} h(t_1, x_1) \\ \vdots \\ h(t_{N_b}, x_{N_b}) \end{bmatrix} \tag{4.26a}$$

$$\text{Periodic:} \quad R^b = \begin{bmatrix} \zeta^T(t_1, \bar{x}_1) - \zeta^T(t_1, \underline{x}_1) \\ \vdots \\ \zeta^T(t_{N_b}, \bar{x}_{N_b}) - \zeta^T(t_{N_b}, \underline{x}_{N_b}) \end{bmatrix}, \quad Y^b = \begin{bmatrix} O_{q,1} \\ \vdots \\ O_{q,1} \end{bmatrix} \tag{4.26b}$$

$$\text{Neumann:} \quad R^b = \begin{bmatrix} \partial_n \zeta^T(t_1, x_1) \\ \vdots \\ \partial_n \zeta^T(t_{N_b}, x_{N_b}) \end{bmatrix}, \quad Y^b = \begin{bmatrix} h(t_1, x_1) \\ \vdots \\ h(t_{N_b}, x_{N_b}) \end{bmatrix} \tag{4.26c}$$

4.2.2.4 Optimal QPINN Weights

The QPINN loss function (4.2) is rewritten as

$$\begin{aligned} \mathcal{L} =& \frac{\gamma_p}{N_p} \sum_{m=1}^{M} \left(\left(Y_m^p - R_m^p \theta\right)^T \left(Y_m^p - R_m^p \theta\right) \right) + \frac{\gamma_i}{N_i} \left(Y^i - R^i \theta\right)^T \left(Y^i - R^i \theta\right) \\ &+ \frac{\gamma_b}{N_b} \left(Y^b - R^b \theta\right)^T \left(Y^b - R^b \theta\right) + \beta \theta^T \theta \end{aligned} \tag{4.27}$$

where $\beta \theta^T \theta$ is a regularization term modulated by $\beta > 0$.

Theorem 3. *The QPINN training problem of the form (2.11), with input-output expression given by (4.15), using a physics-informed loss function given by (4.27) has optimal weights given by*

$$\begin{aligned} \theta &= \Gamma_1^{-1} \Gamma_2 \\ \Gamma_1 &= \frac{\gamma_p}{N_p} \sum_{m=1}^{M} (R_m^p)^T R_m^p + \frac{\gamma_i}{N_i} (R^i)^T R^i + \frac{\gamma_b}{N_b} (R^b)^T R^b + \beta I \\ \Gamma_2 &= \frac{\gamma_p}{N_p} \sum_{m=1}^{M} (R_m^p)^T Y_m^p + \frac{\gamma_i}{N_i} (R^i)^T Y^i + \frac{\gamma_b}{N_b} (R^b)^T Y^b \end{aligned} \tag{4.28}$$

where $\gamma_{pde} \geq 0$, $\gamma_{ic} \geq 0$, $\gamma_{bc} \geq 0$, *and* $\beta > 0$.

Proof. The QPINN loss function can be expanded as

$$\begin{aligned} \mathcal{L} =& \frac{\gamma_p}{N_p} \sum_{m=1}^{M} \left((Y_m^p)^T Y_m^p - 2 (Y_m^p)^T R_m^p \theta + \theta^T (R_m^p)^T R_m^p \theta \right) \\ &+ \frac{\gamma_i}{N_i} \left((Y^i)^T Y^i - 2 (Y^i)^T R^i \theta + \theta^T (R^i)^T R^i \theta \right) \\ &+ \frac{\gamma_b}{N_b} \left((Y^b)^T Y^b - 2 (Y^b)^T R^b \theta + \theta^T (R^b)^T R^b \theta \right) + \beta \theta^T \theta \end{aligned}$$

given that $Y^T R \theta$ is a scalar, which makes $Y^T R \theta = \theta^T R^T Y$ for each Y, R pair. Taking the

first derivative to check the necessary condition for optimality yields

$$\begin{aligned}\partial_\theta \mathcal{L} =& \frac{\gamma_p}{N_p} \sum_{m=1}^{M} \left(-2(Y_m^p)^T R_m^p + 2(R_m^p)^T R_m^p \theta\right) \\ &+ \frac{\gamma_i}{N_i} \left(-2(Y^i)^T R^i + 2(R^i)^T R^i \theta\right) \\ &+ \frac{\gamma_b}{N_b} \left(-2(Y^b)^T R^b + 2(R^b)^T R^b \theta\right) + 2\beta\theta = 0 \\ \iff& \theta = \Gamma_1^{-1} \Gamma_2\end{aligned}$$

with Γ_1 and Γ_2 defined in (4.28). Verifying the sufficient condition with the second derivative shows that it is minimized, i.e.,

$$\partial_{\theta\theta} \mathcal{L} = \frac{\gamma_p}{N_p} \sum_{m=1}^{M} \left(2(R_m^p)^T R_m^p\right) + \frac{\gamma_i}{N_i} \left(2(R^i)^T R^i\right) + \frac{\gamma_b}{N_b} \left(2(R^b)^T R^b\right) + 2\beta I$$

$$\beta > 0,\ \gamma_p \geq 0,\ \gamma_i \geq 0,\ \gamma_b \geq 0 \implies \partial_{\theta\theta} \mathcal{L} \succ 0$$

□

Remark 9 (QPINN regularization)**.** *Theorem 3 requires that $\beta \neq 0$ such that $\partial_{\theta\theta} \mathcal{L} \succ 0$, which also ensures the invertibility of Γ_1. The importance of the regularization term will be seen in Section 4.5.*

4.3 PDE Symmetries

The following section provides a brief introduction to symmetry analysis with applications to differential equations. A more comprehensive overview of the topic, along with proofs of the results, can be found in [53] and [54].

4.3.1 Symmetries and Lie Groups

Symmetries are transformations that, when applied to an object, leave the object invariant. Let us first define symmetries in the context of functions. Consider a function $f(x, y)$ with

scalar inputs x and y. Let $\tilde{x} = g(x, y, a)$ and $\tilde{y} = h(x, y, a)$ be transformations defined by some parameter a with the property

$$\tilde{x}|_{a=0} = g(x, y, 0) = x, \quad \tilde{y}|_{a=0} = h(x, y, 0) = y \tag{4.29}$$

Since these transformations are defined by a single parameter, we call them one-parameter transformations. If $f(\tilde{x}, \tilde{y}) = f(x, y)$, then the function is said to be invariant under the transformation, and we call the transformation a symmetry of the function $f(x, y)$.

Example 11 (Synmmetry of a function)**.** *[55] Consider the function $f(x, y) = xy^{-1}$, define the one-parameter transformation $\tilde{x} = xe^a$ and $\tilde{y} = ye^a$. It can be seen that $f(x, y)$ is invariant under the transformation, i.e.,*

$$f(\tilde{x}, \tilde{y}) = \tilde{x}\tilde{y}^{-1} = (xe^a)(ye^a)^{-1} = xy^{-1} = f(x, y)$$

In the context of PDEs, symmetries are transformations, that, when applied to the dependent and independent variables of the system, transform a solution into another.

Example 12 (Symmetry of a PDE)**.** *[55] Consider the heat PDE given by $\partial_t u - \nu \partial_{xx} u = 0$, define the one-parameter transformation: $\tilde{x} = xe^a$, $\tilde{t} = te^{2a}$, and $\tilde{u} = u$. If u is a solution to the PDE in the space (t, x), then $\tilde{u}$ in the space $(\tilde{x}, \tilde{t})$ is also a solution since*

$$\partial_{\tilde{t}} \tilde{u} - \nu \partial_{\tilde{x}\tilde{x}} \tilde{u} = (e^{2a})^{-1} \partial_t u - \nu (e^a)^{-2} \partial_{xx} u = e^{-2a}(\partial_t u - \nu \partial_{xx} u) = e^{-2a}(0) = 0$$

A collection of these one-parameter transformations that leave an object invariant form a group[1]. Specifically, we consider groups of continuously differentiable symmetries. Formally, these are called Lie groups, as they are differentiable manifolds.

[1]Formally, groups are required to satisfy the following properties: closure, associativity, existence of an identity, and existence of an inverse. For a more complete definition of groups, see [53].

4.3.2 Infinitesimal Generators

Since we are dealing with continuously differentiable symmetries, we can consider infinitesimal transformations, i.e., $\tilde{x} = g(x, y, a + \delta a)$ and $\tilde{y} = h(x, y, a + \delta a)$, where δa is an infinitesimal change in a. Taking the Taylor series expansion of $g(x, y, a + \delta a)$ and $h(x, y, a + \delta a)$ around the point $a = 0$ yields

$$\begin{aligned} \tilde{x} &= g(x, y, 0) + (\partial_a g)|_{a=0}\, \delta a + 0.5\, (\partial_{aa} g)|_{a=0}\, (\delta a)^2 + \cdots \\ \tilde{y} &= h(x, y, 0) + (\partial_a h)|_{a=0}\, \delta a + 0.5\, (\partial_{aa} h)|_{a=0}\, (\delta a)^2 + \cdots \end{aligned} \tag{4.30}$$

Truncating (4.30) to the 1st order approximation and using property (4.29) yields

$$\begin{aligned} \tilde{x} - x = \delta x &= (\partial_a g)|_{a=0}\, \delta a = \xi_x(x, y)\delta a \\ \tilde{y} - y = \delta y &= (\partial_a h)|_{a=0}\, \delta a = \xi_y(x, y)\delta a \end{aligned} \tag{4.31}$$

where $\xi_x(x, y) = (\partial_a g)|_{a=0}$ and $\xi_y(x, y) = (\partial_a h)|_{a=0}$. The infinitesimal change of the function f subject to the transformation at the point (x^*, y^*) is given by the chain rule as the Lie Series

$$f(\tilde{x}, \tilde{y})|_{x^*, y^*} = f(x^* + \delta x, y^* + \delta y) = f(x^*, y^*) + (\xi_x \partial_x f + \xi_y \partial_y f)|_{x^*, y^*}\, \delta_a + \cdots \tag{4.32}$$

Truncating (4.32) to the 1st order approximation yields

$$f(x^* + \delta x, y^* + \delta y) - f(x^*, y^*) = \delta f|_{x^*, y^*} = (\xi_x \partial_x f + \xi_y \partial_y f)|_{x^*, y^*}\, \delta_a \tag{4.33}$$

Defining the differential operator $v = \xi_x \partial_x + \xi_y \partial_y$, equation (4.33) can be rewritten as

$$\delta f|_{x^*, y^*} = v[f]|_{x^*, y^*}\, \delta_a \tag{4.34}$$

were $v[f]$ is v applied to the function f. We call v the infinitesimal generator of the symmetry, as it is a vector field which generates the flow of the transformation at a given point. The terms $\xi_x = \xi_x(x, y)$ and $\xi_y = \xi_y(x, y)$ are the infinitesimal generator coefficients.

Example 13 (Infinitesimal generator)**.** *Consider the function $f(x, y) = xy^{-1}$ with a symmetry given by transformation $\tilde{x} = xe^a$, $\tilde{y} = ye^a$. Substituting $\tilde{x}$ and $\tilde{y}$ in equation (4.31) yields*

$$\delta x = (xe^a)|_{a=0}\,\delta a = x\delta a \implies \xi_x = x$$

$$\delta y = (ye^a)|_{a=0}\,\delta a = y\delta a \implies \xi_y = y$$

which gives the infinitesimal generator $v = x\partial_x + y\partial_y$.

PDE symmetries also have infinitesimal generators. For a system with an independent temporal variable t, p independent spatial variables $x_1, \ldots, x_p$, and q dependent variables $u_1, \ldots, u_\alpha$, the infinitesimal generators are of the form

$$v = \xi_t \partial_t + \sum_{i=1}^{p} \xi_{x_i} \partial_{x_i} + \sum_{\alpha=1}^{q} \phi_\alpha \partial_{u_\alpha} \tag{4.35}$$

where ξ_t, ξ_{x_i}, and ϕ_α are the infinitesimal generator coefficients associated with the temporal variable t, the i^{th} spatial variable x_i, and the α^{th} dependent variable u_α, respectively. Each Lie group of symmetries has an associated Lie algebra, which is the set of vector fields that describe the infinitesimal symmetry transformations of the Lie group [38].

Example 14 (Infinitesimal generators of the heat equation)**.** *[53] The heat equation's symmetry group Lie algebra is spanned by the following infinitesimal generators*

$$v_1 = \partial_x, \qquad v_2 = \partial_t, \qquad v_3 = u\partial_u,$$

$$v_4 = x\partial_x + 2t\partial_t, \qquad v_5 = 2\nu t\partial_x - xu\partial_u, \qquad v_6 = 4\nu t x\partial_x + 4\nu t^2\partial_t - (x^2 + 2\nu t)u\partial_u$$

where v_1 and v_2 correspond to spatial and temporal translation symmetry, and v_4 corresponds to the dilation symmetry (see Example 12).

4.3.3 Prolongation

Given a vector of dependent variables $u \in \mathbb{R}^q$, the vector $u^{(n)} \in \mathbb{R}^{qp^{(n)}}$ is the n^{th} order prolongation of u, which is an extended vector containing u and all of its derivatives up to the n^{th} order with respect to the independent variables $t \in \mathbb{R}$ and $x \in \mathbb{R}^p$. The prolonged vector $u^{(n)}$ is of length $qp^{(n)}$ where

$$qp^{(n)} \equiv q \begin{pmatrix} p+n+1 \\ n \end{pmatrix} = q\frac{(p+n+1)(p+n)\cdots(p+2)}{n(n-1)\cdots 1} \tag{4.36}$$

Consider a space whose coordinates are given by the dependent and independent variables of a PDE (t, x, u). The n^{th} order prolongation of the space is given by $(t, x, u^{(n)})$. In this prolonged space, differential equations can be viewed as algebraic equations whose solutions satisfy $\Delta(t, x, u^{(n)}) = 0$. As an example, the heat equation is given by $\Delta(t, x, u^{(2)}) = \partial_t u - \nu\partial_{xx}u = 0$, with $u^{(2)} = (u, \partial_t u, \partial_x u, \partial_{tt}u, \partial_{xx}u, \partial_{tx}u)$. This prolongation can also be applied to the infinitesimal generators, which yields [53]

$$v^{(n)} = \xi_t\partial_t + \sum_{i=1}^{p}\xi_{x_i}\partial_{x_i} + \sum_{\alpha=1}^{q}\sum_{J}\phi_\alpha^J\partial_{\partial_J u_\alpha} \tag{4.37}$$

where J is all unordered monomials of $\{t, x_1, \ldots, x_p\}$ of degree 0 to n (e.g. $n = 2$, $p = 2$ gives $J = 1, t, x_1, x_2, tt, x_1x_1, x_2x_2, tx_1, tx_2, x_1x_2$), $\partial_{\partial_J u_\alpha}$ is the partial derivative with respect to $\partial_J u_\alpha$, and [53]

$$\phi_\alpha^J = D_J\left[\phi_\alpha - \xi_t\partial_t u_\alpha - \sum_{i=1}^{p}\xi_{x_i}\partial_{x_i}u_\alpha\right] + \xi_t\partial_t\partial_J u_\alpha + \sum_{i=1}^{p}\xi_{x_i}\partial_{x_i}\partial_J u_\alpha \tag{4.38}$$

where D_J is the total derivative operator. The following lemma will be used in the definition of the symmetry loss.

Lemma 8. *[38] For an infinitesimal generator v of a symmetry of the PDE $\Delta(t, x, u^{(n)})$, if $\Delta = 0$ then $v^{(n)}[\Delta] = 0$.*

Example 15 (Prolongation of heat equation symmetry)**.** *[38] Consider the infinitesimal generator* $v = 2\nu t\partial_x - xu\partial_u$ *for one of the symmetries of the heat equation* $\Delta = \partial_t u - \nu\partial_{xx}u = 0$*. Since* Δ *is* 2^{nd} *order we consider the* 2^{nd} *order prolongation of* v*, which, since* u *and* x *are scalars (*$q = 1$*,* $p = 1$*), is given by equation (4.37) as*

$$\begin{aligned} v^{(2)} &= \xi_t\partial_t + \sum_{i=1}^{p}\xi_{x_i}\partial_{x_i} + \sum_{\alpha=1}^{q}\sum_{J}\phi_\alpha^J\partial_{\partial_J u_\alpha} \\ &= \xi_t\partial_t + \xi_x\partial_x + \sum_{J}\phi^J\partial_{\partial_J u} \end{aligned}$$

Since it is the 2^{th} *order prolongation,* $J = 1, t, x, tt, xx, tx$*, which yields*

$$v^{(2)} = \xi_t\partial_t + \xi_x\partial_x + \phi\partial_u + \phi^t\partial_{\partial_t u} + \phi^x\partial_{\partial_x u} + \phi^{tt}\partial_{\partial_{tt}u} + \phi^{xx}\partial_{\partial_{xx}u} + \phi^{tx}\partial_{\partial_{tx}u} \tag{4.39}$$

Applying (4.39) to Δ *yields*

$$\begin{aligned} v^{(2)}[\Delta] &= \xi_t\partial_t\Delta + \xi_x\partial_x\Delta + \phi\partial_u\Delta + \phi^t\partial_{\partial_t u}\Delta + \phi^x\partial_{\partial_x u}\Delta + \phi^{tt}\partial_{\partial_{tt}u}\Delta + \phi^{xx}\partial_{\partial_{xx}u}\Delta + \phi^{tx}\partial_{\partial_{tx}u}\Delta \\ &= \xi_t(0) + \xi_x(0) + \phi(0) + \phi^t(1) + \phi^x(0) + \phi^{tt}(0) + \phi^{xx}(-\nu) + \phi^{tx}(0) \\ &= \phi^t - \nu\phi^{xx} \end{aligned} \tag{4.40}$$

Computing the infinitesimal generator coefficient ϕ^t *in (4.40) using (4.38) and with* $q = 1$*,* $p = 1$ *gives*

$$\begin{aligned} \phi^t &= D_J\left[\phi_\alpha - \xi_t\partial_t u_\alpha - \sum_{i=1}^{p}\xi_i\partial_{x_i}u_\alpha\right] + \xi_t\partial_t\partial_J u_\alpha + \sum_{i=1}^{p}\xi_{x_i}\partial_{x_i}\partial_J u_\alpha \\ &= D_t\left[\phi - \xi_t\partial_t u - \xi_x\partial_x u\right] + \xi_t\partial_{tt}u + \xi_x\partial_{tx}u \end{aligned}$$

Taking the derivative using the chain rule yields

$$\begin{aligned} \phi^t &= \left[\partial_t\phi - (\partial_t\xi_t)\partial_t u - \xi_t\partial_{tt}u - (\partial_t\xi_x)\partial_x u - \xi_x\partial_{tx}u\right] + \xi_t\partial_{tt}u + \xi_x\partial_{tx}u \\ &= \partial_t\phi - (\partial_t\xi_t)\partial_t u - (\partial_t\xi_x)\partial_x u \end{aligned}$$

Finally, substituting in $\phi = -xu$, $\xi_t = 0$, *and* $\xi_x = 2\nu t$ *obtained from* v *yields*

$$\begin{aligned}\phi^t &= \partial_t(-xu) - (\partial_t(0))\partial_t u - (\partial_t(2\nu t))\partial_x u \\ &= -x\partial_t u - 2\nu\partial_x u\end{aligned}$$

Applying the same process to calculate ϕ^{xx} *gives*

$$\begin{aligned}\phi^{xx} &= D_J\left[\phi_\alpha - \xi_t\partial_t u_\alpha - \sum_{i=1}^{p}\xi_i\partial_{x_i}u_\alpha\right] + \xi_t\partial_t\partial_J u_\alpha + \sum_{i=1}^{p}\xi_{x_i}\partial_{x_i}\partial_J u_\alpha \\ &= D_{xx}\left[\phi - \xi_t\partial_t u - \xi_x\partial_x u\right] + \xi_t\partial_{txx}u + \xi_x\partial_{xxx}u \\ &= \partial_{xx}\phi - (\partial_{xx}\xi_t)\partial_t u - 2(\partial_x\xi_t)\partial_{tx}u - (\partial_{xx}\xi_x)\partial_x u - 2(\partial_x\xi_x)\partial_{xx}u \\ &= -2\partial_x u - x\partial_{xx}u\end{aligned}$$

Substituting $\phi^t = -x\partial_t u - 2\nu\partial_x u$ *and* $\phi^{xx} = -2\partial_x u - x\partial_{xx}u$ *into (4.40) yields*

$$v^{(2)}[\Delta] = \phi^t - \nu\phi^{xx} = (-x\partial_t u - 2\nu\partial_x u) - \nu(-2\partial_x u - x\partial_{xx}u) = -x(\partial_t u - \nu\partial_{xx}u) = -x\Delta$$

where it can be seen that $\Delta = 0 \implies v^{(2)}[\Delta] = 0$, *as stated in Lemma 8.*

4.3.4 Symmetry Loss

The integration of PDE symmetries into the PINN loss function was proposed for the first time in [38]. To accomplish this, the PINN loss function presented in (4.2) is modified as

$$\mathcal{L} = \gamma_{pde}\mathcal{L}_{pde} + \gamma_{ic}\mathcal{L}_{ic} + \gamma_{bc}\mathcal{L}_{bc} + \gamma_{sym}\mathcal{L}_{sym} + \beta\mathcal{L}_{reg} \tag{4.41}$$

where $\gamma_{sym} \geq 0$. The symmetry loss $\mathcal{L}_{sym}$ is given by [38]

$$\mathcal{L}_{sym} = \frac{1}{N_s}\sum_{k=1}^{K}\sum_{i=1}^{N_s}\| J_\Delta(t_i, x_i)\mathbf{coef}\left(v_k^{(n)}(t_i, x_i)\right)\|_2^2 \tag{4.42}$$

where K is the number of symmertries and $\{(t_i, x_i)\}_{i=1}^{N_s}$ are N_s points sampled for $t \in [0, T]$ and $x \in \Omega$. Furthermore, since $\Delta = [\Delta_1 \ \cdots \ \Delta_M]^T$ is a set of M PDEs, J_Δ is the Jacobian matrix of Δ with respect to the independent and prolonged dependent variables given by

$$J_\Delta = \begin{bmatrix} J_{\Delta_1} \\ \vdots \\ J_{\Delta_M} \end{bmatrix} = \begin{bmatrix} \partial_t \Delta_1 & (\partial_x \Delta_1)^T & (\partial_{u^{(n)}} \Delta_1)^T \\ \vdots & \vdots & \vdots \\ \partial_t \Delta_M & (\partial_x \Delta_M)^T & (\partial_{u^{(n)}} \Delta_M)^T \end{bmatrix} \in \mathbb{R}^{M \times (1+p+qp^{(n)})} \tag{4.43}$$

and $\mathbf{coef}\left(v_k^{(n)}\right)$ is the vector containing the coefficients of the prolonged infinitesimal generator $v_k^{(n)}$ given as

$$\mathbf{coef}\left(v_k^{(n)}\right) = \begin{bmatrix} \xi_t & \xi_x^T & (\phi^{(n)})^T \end{bmatrix}^T \in \mathbb{R}^{1+p+qp^{(n)}} \tag{4.44}$$

where $\xi_x \in \mathbb{R}^p$ and $\phi^{(n)} \in \mathbb{R}^{qp^{(n)}}$ are vectors containing the infinitesimal generator coefficients of (4.37) associated with the partial derivatives with respect to x and $u^{(n)}$, respectively. From (4.43) and (4.44) it follows that

$$\begin{aligned} J_\Delta \mathbf{coef}\left(v_k^{(n)}\right) &= \begin{bmatrix} \partial_t \Delta_1 & (\partial_x \Delta_1)^T & (\partial_{u^{(n)}} \Delta_1)^T \\ \vdots & \vdots & \vdots \\ \partial_t \Delta_M & (\partial_x \Delta_M)^T & (\partial_{u^{(n)}} \Delta_M)^T \end{bmatrix} \begin{bmatrix} \xi_t \\ \xi_x \\ \phi^{(n)} \end{bmatrix} \\ &= \begin{bmatrix} (\partial_t \Delta_1)\xi_t + (\partial_x \Delta_1)^T \xi_x + (\partial_{u^{(n)}} \Delta_1)^T \phi^{(n)} \\ \vdots \\ (\partial_t \Delta_M)\xi_t + (\partial_x \Delta_M)^T \xi_x + (\partial_{u^{(n)}} \Delta_M)^T \phi^{(n)} \end{bmatrix} \end{aligned} \tag{4.45}$$

Since each Δ_m is scalar-valued,

$$\begin{aligned} J_\Delta \mathbf{coef}\left(v_k^{(n)}\right) &= \begin{bmatrix} [(\partial_t)\xi_t + (\partial_x)^T \xi_x + (\partial_{u^{(n)}})^T \phi^{(n)}]\Delta_1 \\ \vdots \\ [(\partial_t)\xi_t + (\partial_x)^T \xi_x + (\partial_{u^{(n)}})^T \phi^{(n)}]\Delta_M \end{bmatrix} \\ &= \left[(\partial_t)\xi_t + (\partial_x)^T \xi_x + (\partial_{u^{(n)}})^T \phi^{(n)}\right] [\Delta_1 \ \cdots \ \Delta_M]^T \end{aligned} \tag{4.46}$$

Since $\Delta = [\Delta_1 \ \cdots \ \Delta_M]^T$ and using (4.37), (4.46) can be written as

$$J_\Delta \mathbf{coef}\left(v_k^{(n)}\right) = \left[(\partial_t)\xi_t + (\partial_x)^T\xi_x + (\partial_{u^{(n)}})^T\phi^{(n)}\right]\Delta = v_k^{(n)}[\Delta] \tag{4.47}$$

Therefore, it can be seen that the symmetry loss term $\mathcal{L}_{sym}$ penalizes solutions for which $v^{(n)}[\Delta] \neq 0$, which from Lemma 8, corresponds to $\Delta \neq 0$.

Example 16 ($J_\Delta \mathbf{coef}\left(v_k^{(n)}\right)$ for the heat equation)**.** *Consider the heat equation given by* $\Delta = \partial_t u - \nu\partial_{xx}u = 0$. *Since* Δ *is a* 2^{nd} *order PDE,* $u^{(2)} = [u \ \partial_t u \ \partial_x u \ \partial_{tt}u \ \partial_{xx}u \ \partial_{tx}u]^T$. *Consider the symmetry given by the infinitesimal generator* $v = 2\nu t\partial_x - xu\partial_u$ *which has* $\xi_t = 0$, $\xi_x = 2\nu t$, *and* $\phi = -xu$. *From (4.44),* $\boldsymbol{coef}\left(v_k^{(n)}\right)$ *is given by*

$$\boldsymbol{coef}\left(v_k^{(n)}\right) = \begin{bmatrix} \xi_t & \xi_x & \phi & \phi^t & \phi^x & \phi^{tt} & \phi^{xx} & \phi^{tx} \end{bmatrix}^T \tag{4.48}$$

where, using (4.38) and following the same process as for ϕ^t *and* ϕ^{xx} *from Example 15, the infinitesimal generator coefficients are*

$$\phi^t = -x\partial_t u - 2\nu\partial_x u, \quad \phi^x = -u - x\partial_x u$$
$$\phi^{tt} = -x\partial_{tt}u - 4\nu\partial_{tx}u, \quad \phi^{xx} = -2\partial_x u - x\partial_{xx}u, \quad \phi^{tx} = -\partial_t u - x\partial_{tx}u - 2\nu\partial_{xx}u$$

From (4.43), since $\partial_{u^{(n)}} = [\partial_u \ \partial_{\partial_t u} \ \partial_{\partial_x u} \ \partial_{\partial_{tt}u} \ \partial_{\partial_{xx}u} \ \partial_{\partial_{tx}u}]^T$, J_Δ *is given by*

$$\begin{aligned} J_\Delta &= \begin{bmatrix} \partial_t\Delta & \partial_x\Delta & \partial_u\Delta & \partial_{\partial_t u}\Delta & \partial_{\partial_x u}\Delta & \partial_{\partial_{tt}u}\Delta & \partial_{\partial_{xx}u}\Delta & \partial_{\partial_{tx}u}\Delta \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & -\nu & 0 \end{bmatrix} \end{aligned} \tag{4.49}$$

Therefore, from (4.48) and (4.49),

$$J_\Delta \boldsymbol{coef}\left(v_k^{(n)}\right) = \phi^t - \nu\phi^{xx} = -x(\partial_t u - \nu\partial_{xx}u) = -x\Delta$$

which is the same result as the one in Example 15.

4.4 Least Squares Symmetry Loss for QPINNs

This section will seek to integrate the symmetry loss proposed by the authors of [38] into the least squares QPINNs training methodology. For this section, it will be useful to consider the linear differential operator $\mathcal{D}^{(n)}[\cdot]$ acting on $u(t,x)$ as a matrix product given as

$$\mathcal{D}^{(n)}[u(t,x)] = \mathbb{D}(t,x)u^{(n)} \tag{4.50}$$

where $\mathbb{D}(t,x) \in \mathbb{R}^{M\times qp^{(n)}}$ is a matrix of the differential equation coefficients for a system of M equations of order n.

Example 17 (Matrix representation of $\mathcal{D}$)**.** *Consider the 1^{st} order system of PDEs of the form (4.1), given by*

$$\Delta = \mathcal{D}^{(1)}[u(t,x)] = \begin{bmatrix} \partial_t u + x_1 \partial_{x_2} u \\ \partial_t u + x_2 \partial_{x_1} u \end{bmatrix} = 0$$

using $u^{(1)} = [u, \partial_t u, \partial_{x_1} u, \partial_{x_2} u]$, the differential operator can be expressed as

$$\mathcal{D}^{(1)}[u(t,x)] = \mathbb{D}u^{(1)} = \begin{bmatrix} 0 & 1 & 0 & x_1 \\ 0 & 1 & x_2 & 0 \end{bmatrix} \begin{bmatrix} u & \partial_t u & \partial_{x_1} u & \partial_{x_2} u \end{bmatrix}^T$$

4.4.1 Symmetry Loss Term

The symmetry loss term for a set of M differential equations with K symmetries with infinitesimal generators v_k is given by

$$\begin{aligned} \mathcal{L}_{sym} &= \frac{1}{N_s} \sum_{k=1}^{K} \sum_{i=1}^{N_s} \| J_\Delta(t_i, x_i) \mathbf{coef}\left(v_k^{(n)}(t_i, x_i) \right) \|_2^2 \\ &= \frac{1}{N_s} \sum_{k=1}^{K} \sum_{i=1}^{N_s} \sum_{m=1}^{M} \left(J_{\Delta_m}(t_i, x_i) \mathbf{coef}\left(v_k^{(n)}(t_i, x_i) \right) \right)^2 \end{aligned} \tag{4.51}$$

where J_Δ and $\mathbf{coef}\left(v_k^{(n)}\right)$ are given by (4.43) and (4.44), respectively. To add the symmetry loss to the least squares training methodology presented in the previous sections, it must be shown that $J_{\Delta_m}\mathbf{coef}\left(v_k^{(n)}\right)$ is affine with respect to $u^{(n)}$ forall $m = 1, \ldots, M$, where J_{Δ_m} is the m^{th} row of J_Δ. This is done in the following Lemma.

Lemma 9. *For a PDE of the form $\Delta_m = \mathbb{D}_m(t,x)u^{(n)} + f_m(t,x) = 0$ with infinitesimal generator v_k, if the infinitesimal generator coefficients are of the form*

$$\xi_t = \xi_t(t,x), \quad \xi_x = \xi_x(t,x) \tag{4.52}$$

$$\phi^{(n)} = \phi_u^{(n)}(t,x)u^{(n)} + \phi_0^{(n)}(t,x) \tag{4.53}$$

then $J_{\Delta_m}\boldsymbol{coef}\left(v_k^{(n)}\right)$ is affine with respect to $u^{(n)}$ and can be written as

$$J_{\Delta_m}\boldsymbol{coef}\left(v_k^{(n)}\right) = \left[\xi_t\partial_t\mathbb{D}_m + \xi_x^T\partial_x\mathbb{D}_m + \mathbb{D}_m\phi_u^{(n)}\right]u^{(n)} + \left[\xi_t\partial_t f_m + \xi_x^T\partial_x f_m + \mathbb{D}_m\phi_0^{(n)}\right] \tag{4.54}$$

Proof. The components of J_{Δ_m} are given as

$$\begin{aligned}
\partial_t\Delta_m &= \partial_t\left(\mathcal{D}_m[u(t,x)] + f_m(t,x)\right) = (\partial_t\mathbb{D}_m)u^{(n)} + \partial_t f_m(t,x) \\
\partial_x\Delta_m &= \partial_x(\mathcal{D}_m[u(t,x)] + f_m(t,x)) = (\partial_x\mathbb{D}_m)u^{(n)} + \partial_x f_m(t,x) \\
\partial_{u^{(n)}}\Delta_m &= \partial_{u^{(n)}}\mathcal{D}_m[u(t,x)] = \partial_{u^{(n)}}(\mathbb{D}_m u^{(n)}) = \mathbb{D}_m^T
\end{aligned} \tag{4.55}$$

Substituting (4.55) into $J_{\Delta_m}\mathbf{coef}\left(v_k^{(n)}\right)$ yields

$$\begin{aligned}
J_{\Delta_m}\mathbf{coef}\left(v_k^{(n)}\right) &= \left((\partial_t\mathbb{D}_m)u^{(n)} + \partial_t f_m\right)^T\xi_t + \left((\partial_x\mathbb{D}_m)u^{(n)} + \partial_x f_m\right)^T\xi_x + \mathbb{D}_m\phi^{(n)} \\
&= (u^{(n)})^T\left((\partial_t\mathbb{D}_m)^T\xi_t + (\partial_x\mathbb{D}_m)^T\xi_x\right) + (\partial_t f_m\xi_t + (\partial_x f_m)^T\xi_x) + \mathbb{D}_m\phi^{(n)}
\end{aligned}$$

By using the properties of the transpose, the equation can be re-written as

$$J_{\Delta_m}\mathbf{coef}\left(v_k^{(n)}\right) = \left(\xi_t\partial_t\mathbb{D}_m + \xi_x^T\partial_x\mathbb{D}_m\right)u^{(n)} + (\xi_t\partial_t f_m + \xi_x^T\partial_x f_m) + \mathbb{D}_m\phi^{(n)} \tag{4.56}$$

Substituting in $\phi^{(n)}$ from equation (4.53) into (4.56), and collecting like terms yields

$$J_{\Delta_m}\mathbf{coef}\left(v_k^{(n)}\right) = \left(\xi_t\partial_t\mathbb{D}_m + \xi_x^T\partial_x\mathbb{D}_m + \mathbb{D}_m\phi_u^{(n)}\right)u^{(n)} + \left(\xi_t\partial_t f_m + \xi_x^T\partial_x f_m + \mathbb{D}_m\phi_0^{(n)}\right)$$

which is affine with respect to $u^{(n)}$ if $\xi_t = \xi_t(t,x)$, $\xi_x = \xi_x(t,x)$. □

Example 18 (Symmetry loss of heat equation)**.** *Consider the 1-dimensional heat equation given by* $\Delta = \partial_t u - \nu\partial_{xx}u = 0$*, for which* $\mathbb{D} = [0\ 1\ 0\ 0\ -\nu\ 0]$ *and* $u^{(2)} = [u\ \partial_t u\ \partial_x u\ \partial_{tt}u\ \partial_{xx}u\ \partial_{tx}u]^T$*. Let us consider the symmetry given by the infinitesimal generator* $v = 2\nu t\partial_x - xu\partial_u$ *which has* $\xi_t = 0$*,* $\xi_x = 2\nu t$*, and* $\phi = -xu$*. We compute the terms of* $\phi^{(2)}$ *using (4.38), which yields*

$$\phi^t = -x\partial_t u - 2\nu\partial_x u, \quad \phi^x = -u - x\partial_x u$$
$$\phi^{tt} = -x\partial_{tt}u - 4\nu\partial_{tx}u, \quad \phi^{xx} = -2\partial_x u - x\partial_{xx}u, \quad \phi^{tx} = -\partial_t u - x\partial_{tx}u - 2\nu\partial_{xx}u$$

from which the affine (or linear in this specific example) expression for $\phi^{(2)}$ *is given by*

$$\phi^{(2)} = \begin{bmatrix}\phi\\ \phi^t\\ \phi^x\\ \phi^{tt}\\ \phi^{xx}\\ \phi^{tx}\end{bmatrix} = \begin{bmatrix}-x & 0 & 0 & 0 & 0 & 0\\ 0 & -x & -2\nu & 0 & 0 & 0\\ -1 & 0 & -x & 0 & 0 & 0\\ 0 & 0 & 0 & -x & 0 & -4\nu\\ 0 & 0 & -2 & 0 & -x & 0\\ 0 & -1 & 0 & 0 & -2\nu & -x\end{bmatrix}\begin{bmatrix}u\\ \partial_t u\\ \partial_x u\\ \partial_{tt}u\\ \partial_{xx}u\\ \partial_{tx}u\end{bmatrix} + \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} = \phi_u^{(2)}u^{(2)} + \phi_0^{(2)}$$

Therefore,

$$\begin{aligned} J_{\Delta_m}\boldsymbol{coef}\left(v_k^{(n)}\right) &= \left(\xi_t\partial_t\mathbb{D}_m + \xi_x^T\partial_x\mathbb{D}_m + \mathbb{D}_m\phi_u^{(n)}\right)u^{(n)} + \left(\xi_t\partial_t f_m + \xi_x^T\partial_x f_m + \mathbb{D}_m\phi_0^{(n)}\right)\\ &= \left(\mathbb{D}_m\phi_u^{(2)}\right)u^{(2)}\\ &= \begin{bmatrix}0 & -x & 0 & 0 & \nu x & 0\end{bmatrix}u^{(2)}\end{aligned}$$

which is linear with respect to $u^{(2)}$*, and is the same result as the one obtained in Example 16.*

Recalling that $\hat{u} = \zeta^T(t,x)\theta$, let the prolonged regressor $\zeta^{(n)}$ be defined as a matrix for which each column corresponds to a derivative of ζ with respect to t and x, matching the corresponding row in $u^{(n)}$. This definition yields

$$\hat{u}^{(n)} = (\zeta^{(n)}(t,x))^T\theta \tag{4.57}$$

Given the conditions (4.52) and (4.53) on ξ_t, ξ_x, and $\phi^{(n)}$, Lemma 9 allows for the symmetry loss term, for K symmetries and M PDEs, to be expressed as

$$\mathcal{L}_{sym} = \frac{1}{N_s}\sum_{k=1}^{K}\sum_{m=1}^{M}\left(Y^s_{r,m} - R^s_{r,m}\theta\right)^T\left(Y^s_{r,m} - R^s_{r,m}\theta\right) \tag{4.58}$$

where

$$R^s_{k,m} = \begin{bmatrix} r^s_{k,m}(t_1,x_1)(\zeta^{(n)}(t_1,x_1))^T \\ \vdots \\ r^s_{k,m}(t_{N_s},x_{N_s})(\zeta^{(n)}(t_{N_s},x_{N_s}))^T \end{bmatrix}, \quad Y^s_{k,m} = \begin{bmatrix} y^s_{k,m}(t_1,x_1) \\ \vdots \\ y^s_{k,m}(t_{N_s},x_{N_s}) \end{bmatrix} \tag{4.59}$$

with

$$\begin{aligned} r^s_{k,m}(t_i,x_i) &= \left[\xi_t\partial_t\mathbb{D}_m + \xi_x^T\partial_x\mathbb{D}_m + \mathbb{D}_m\phi^{(n)}_u\right]\Big|_{t=t_i,x=x_i} \\ y^s_{k,m}(t_i,x_i) &= \left[\xi_t\partial_t f_m + \xi_x^T\partial_x f_m + \mathbb{D}_m\phi^{(n)}_0\right]\Big|_{t=t_i,x=x_i} \end{aligned} \tag{4.60}$$

4.4.2 Optimal QPINN Weights with Symmetries

The complete QPINN loss function with added symmetry term is given as

$$\begin{aligned} \mathcal{L} =& \frac{\gamma_p}{N_p}\sum_{m=1}^{M}\left(Y^p_m - R^p_m\theta\right)^T\left(Y^p_m - R^p_m\theta\right) \\ &+ \frac{\gamma_i}{N_i}\left(Y^i - R^i\theta\right)^T\left(Y^i - R^i\theta\right) + \frac{\gamma_b}{N_b}\left(Y^b - R^b\theta\right)^T\left(Y^b - R^b\theta\right) \\ &+ \frac{\gamma_s}{N_s}\sum_{k=1}^{K}\sum_{m=1}^{M}\left(Y^s_{k,m} - R^s_{k,m}\theta\right)^T\left(Y^s_{k,m} - R^s_{k,m}\theta\right) + \beta\theta^T\theta \end{aligned} \tag{4.61}$$

where $\beta\theta^T\theta$ is a regularization term modulated by $\beta > 0$.

Corollary 3.1 (of Theorem 3)**.** *The QPINN training problem of the form (2.11), with input-output expression given by (4.15), using a physics-informed loss function (4.61) has optimal weights given by*

$$\theta = \Gamma_1^{-1}\Gamma_2$$

$$\Gamma_1 = \frac{\gamma_p}{N_p}\sum_{m=1}^{M}(R_m^p)^T R_m^p + \frac{\gamma_i}{N_i}(R^i)^T R^i + \frac{\gamma_b}{N_b}(R^b)^T R^b + \frac{\gamma_s}{N_s}\sum_{k=1}^{K}\sum_{m=1}^{M}(R_{k,m}^s)^T R_{k,m}^s + \beta I \tag{4.62}$$

$$\Gamma_2 = \frac{\gamma_p}{N_p}\sum_{m=1}^{M}(R_m^p)^T Y_m^p + \frac{\gamma_i}{N_i}(R^i)^T Y^i + \frac{\gamma_b}{N_b}(R^b)^T Y^b + \frac{\gamma_s}{N_s}\sum_{k=1}^{K}\sum_{m=1}^{M}(R_{k,m}^s)^T Y_{k,m}^s$$

where $\gamma_{pde} \geq 0$*,* $\gamma_{ic} \geq 0$*,* $\gamma_{bc} \geq 0$*,* $\gamma_{sym} \geq 0$*, and* $\beta > 0$*.*

Proof. The proof follows the proof of Theorem 3, but with the added term for the symmetry loss. Remark 9 on the necessity of $\beta > 0$ is also applicable to this theorem. □

4.5 Examples

For all examples, the PyTorch library in Python was used for the backpropagation-trained physics-informed neural networks (BP-PINN). The BP-PINN loss function was implemented using the code provided by the authors of the original PINN papers [34, 56, 57]. All BP-PINNs use the Tanh activation function in their hidden layers, and were trained with the Adam algorithm using the default PyTorch settings. All quadratic networks use $a = 0.0937$, $b = 0.5$, $c = 0.4688$ for their activation function parameters following the optimal mean squared error ReLU approximation over the interval $[-5, 5]$, which is presented in Section 5.1. Finally, unless otherwise specified, all loss term weights are unitary, i.e., $\gamma_{pde} = 1$, $\gamma_{ic} = 1$, $\gamma_{bc} = 1$, and $\gamma_{sym} = 1$.

4.5.1 Example 1: Heat Equation

The diffusion of heat in a material is governed by a 2nd order PDE called the heat equation. The 1-dimensional heat equation is

$$\Delta = \partial_t u - \nu \partial_{xx} u = 0 \tag{4.63}$$

where t is time, x is position along the material, $u(t, x)$ is temperature, and $\nu > 0$ is the thermal diffusivity coefficient. For this example, $\nu = 0.05$ was used, along with the bounds, initial condition, and periodic boundary conditions given by

$$\begin{gathered} x \in [0, 2\pi], \quad t \in [0, 20] \\ u(0, x) = \sin(x) + 1 \\ u(t, 0) = u(t, 2\pi) \end{gathered} \tag{4.64}$$

4.5.1.1 Part A: QPINN and BP-PINN Comparison

For this part, a QPINN was trained using the following lifted input vector

$$X_i = \begin{bmatrix} t_i & x_i & \sin t_i & \cos t_i & \sin x_i & \cos x_i \end{bmatrix} \tag{4.65}$$

and with a regularization coefficient $\beta = 10^{-8}$. Furthermore, the QPINN used $N_p = 100$, $N_i = 10$, and $N_b = 10$, with all sampled points selected uniformly over the region. For comparison purposes, a backpropagation-trained PINN (BP-PINN) with 3 hidden layers, each of which contains 10 neurons, was trained over 5k epochs. The BP-PINN used $N_p = 625$, $N_i = 20$, and $N_b = 20$, also uniformly sampled. These hyperparameters for the BP-PINN were selected to provide a similar performance to the QPINN. To act as the true solution, a

finite element method (FEM) solver was implemented in Python using

$$u(x,t+\Delta t) = u(x,t) + \nu \frac{\Delta t}{(\Delta x)^2}\left(u(x-\Delta x,t) + u(x+\Delta x,t) - 2u(x,t)\right)$$
$$u(-\Delta x,t) = u(2\pi,t), \quad u(2\pi+\Delta x,t) = u(0,t) \tag{4.66}$$

where Δx and Δt are the resolution parameters for the grid of FEM nodes. The solution used $\Delta x = 0.02\pi$ and $\Delta t = 0.02$, as it was found to provide stable results.

The FEM, QPINN, and BP-PINN solutions are shown in Figure 4.3, with a comparison of their performances provided in Table 4.1. The BP-PINN and QPINN achieved comparable results for both mean squared error and maximum absolute error. However, the QPINN saw a significant reduction in training time compared to the BP-PINN, while requiring fewer sample points to achieve good results. Finally, the QPINN also saw a reduction in training time compared to the FEM solution, which is an advantage of QPINNs, as typically FEMs have shorter training times compared to BP-PINNs [39].

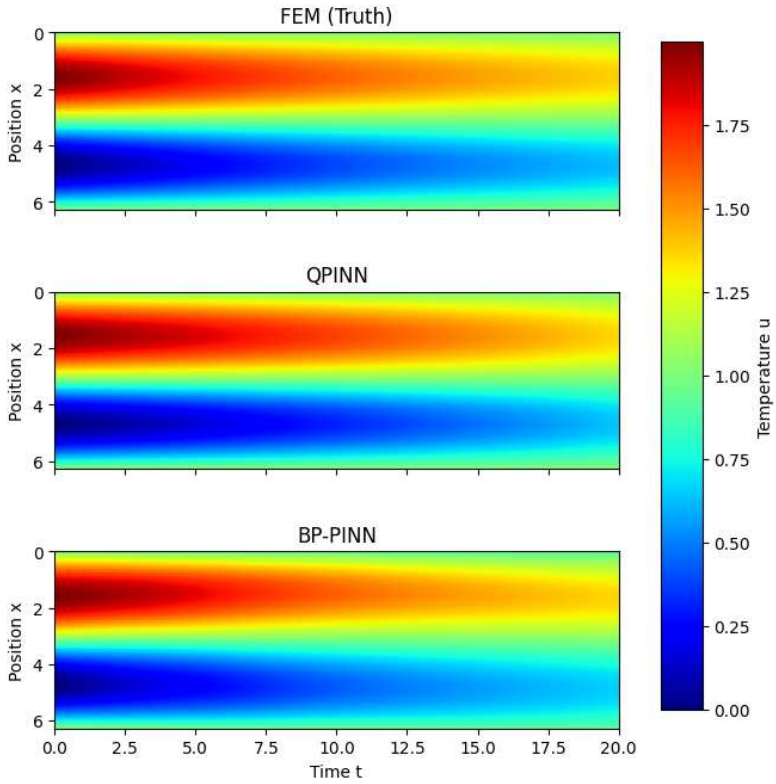


Figure 4.3: FEM, QPINN, and BP-PINN solutions for 1-dimensional heat equation

Table 4.1: Performance comparison between FEM, QPINN, and BP-PINN solutions of 1-dimensional heat equation

Solution	Mean Squared Error	Maximum Absolute Error	Training Time [s]
FEM	n/a	n/a	0.245
QPINN	1.18×10^{-3}	7.06×10^{-2}	0.036
BP-PINN	1.23×10^{-3}	9.85×10^{-2}	39.82

4.5.1.2 Part B: Effects of Regularization

As mentioned in Remark 9, regularization plays an important role in training QPINNs, especially when lifted inputs are used. Multiple QPINNs were trained with various values of the regularization coefficient β. The networks used a lifted input vector X_i whose elements are all the monomials of t and x up to degree 4, i.e.,

$$X_i = \begin{bmatrix} t & x & t^2 & tx & x^2 & \cdots & t^2x^2 & tx^3 & x^4 \end{bmatrix} \tag{4.67}$$

Figure 4.4 shows solutions of QPINNs for three different values of β.

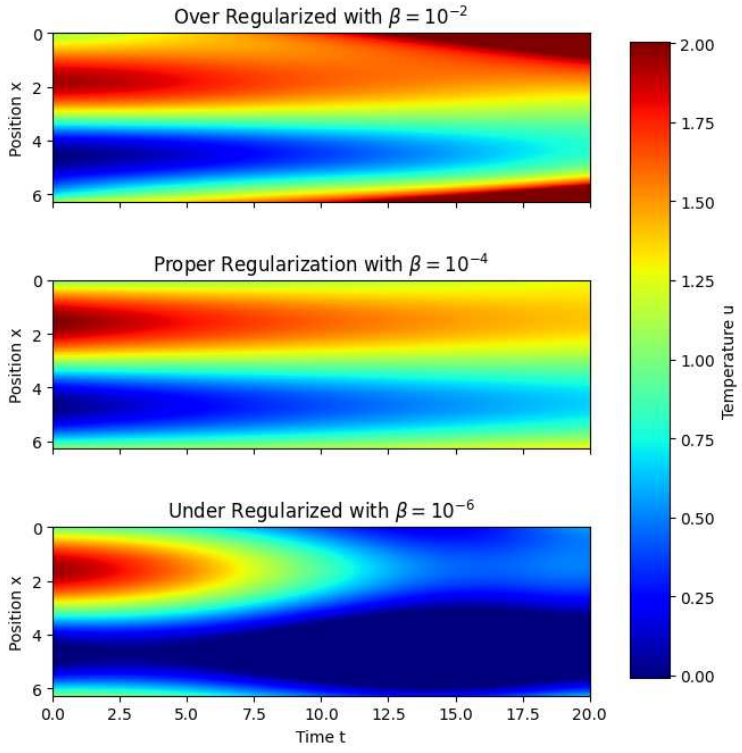


Figure 4.4: Sensitivity of QPINN solution to regularization coefficient β, using monomial lifting for the 1-dimensional heat equation

4.5.1.3 Part C: Effects of Symmetry Loss

For this part, symmetry loss was added to the QPINN training. The complete set of 1-dimensional heat equation symmetries is given as

$$\begin{aligned} &v_1 = \partial_x, && v_2 = \partial_t, && v_3 = u\partial_u, \\ &v_4 = x\partial_x + 2t\partial_t, && v_5 = 2\nu t\partial_x - xu\partial_u, && v_6 = 4\nu tx\partial_x + 4\nu t^2\partial_t - (x^2 + 2\nu t)u\partial_u \end{aligned} \tag{4.68}$$

Not all symmetries produce a non-trivial loss term. For example, the $v_1 = \partial_x$ yields

$$J_\Delta \mathbf{coef}\left(v^{(2)}\right) = 0 \tag{4.69}$$

For this reason, $v_5 = 2\nu t\partial_x - xu\partial_u$ was selected, which yields the non-trivial loss term

$$J_\Delta \mathbf{coef}\left(v_5^{(2)}\right) = x(u_t - \nu u_{xx}) \tag{4.70}$$

The symmetry loss was added to the under regularized case presented in the previous part, with the results being shown in Figure 4.5. It can be seen that adding symmetry loss improves the solution. This result suggests that symmetry loss may act as an additional form of regularization for QPINNs.

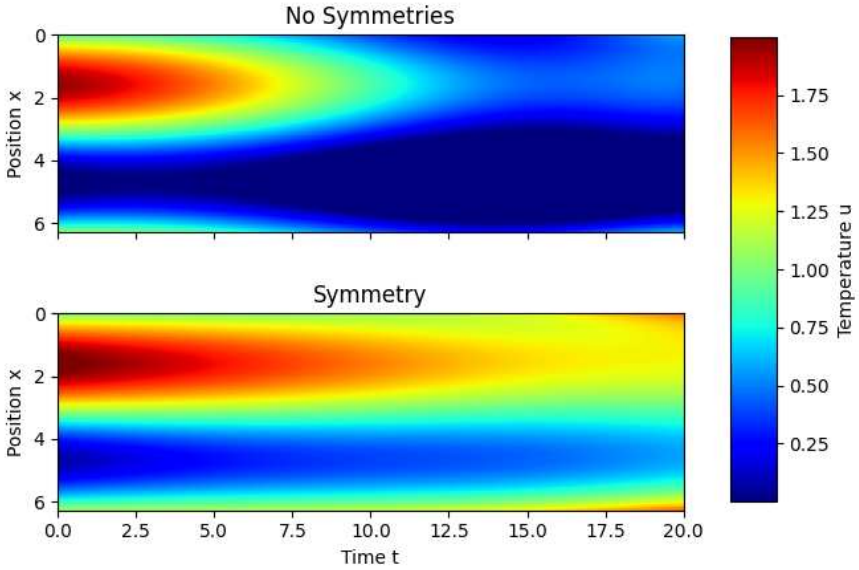


Figure 4.5: Effects of adding symmetry loss to a QPINN with insufficient regularization

4.5.2 Example 2: Optimal Control

Solving optimal control problems requires computing the solution to a partial differential equation known as the Hamilton-Jacobi-Bellman (HJB) equation. The authors of [41] proposed the use of PINNs to solve the HJB in an iterative fashion. For simplicity, this example will showcase the method for a scalar state x and a scalar input u. Consider the optimal control problem

$$\begin{aligned} &\min_{u} \int_0^\infty (qx^2 + ru^2)dt \\ &\text{s.t. } \dot{x} = f(x) + g(x)u \end{aligned} \tag{4.71}$$

The objective is to find the optimal controller $u = \kappa(x)$ which solves (4.71). Starting with an initial stabilizing controller $\kappa_0(x)$, the optimal controller is found over $i \geq 0$ iterations using the following two step procedure [41]:

1. (Policy Evaluation) Compute a value function $V_i(x)$ by solving the PDE

$$(\partial_x V_i(x))(f(x) + g(x)\kappa_i(x)) = -q - r\kappa_i^2(x) \tag{4.72}$$

   subject to $V_i(0) = 0$.

2. (Policy Improvement) Update the control law using

$$\kappa_{i+1}(x) = -\frac{1}{2} r^{-1} g(x)(\partial_x V_i(x)) \tag{4.73}$$

Consider the optimal control problem

$$\begin{aligned} &\min_{u} \int_0^\infty (x^2 + u^2)dt \\ &\text{s.t. } \dot{x} = x^2 + u \end{aligned} \tag{4.74}$$

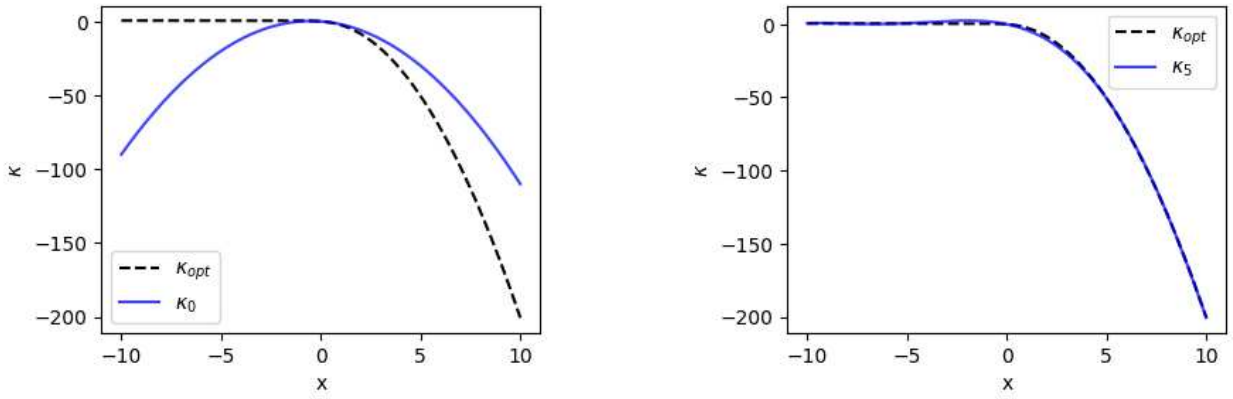
for which $f(x) = x^2$, $g(x) = 1$, $r = 1$, and $q = 1$. Using Pontryagin's Maximum Principle (PMP), the known solution of (4.74) is

$$\kappa_{opt}(x) = -x^2 - x\sqrt{x^2 + 1} \tag{4.75}$$

Using a QPINN, equation (4.72) of the iterative HJB method was solved. The QPINN used a regularization coefficient $\beta = 10^{-5}$, $N_i = 1$, $N_p = 100$, and the lifted input

$$X_i = \begin{bmatrix} x_i & x_i^2 & x_i^3 \end{bmatrix}^T \tag{4.76}$$

Starting from the initial stabilizing controller $\kappa_0(x) = -x^2 - x$ the method was carried out for 5 iterations. The initial controller is shown in Figure 4.6a, and the final controller after 5 iterations is shown in Figure 4.6b. It can be seen that the QPINN successfully converged to the optimal controller.



(a) Initial controller $\kappa_0(x)$

(b) Controller $\kappa_5(x)$ after 5 iterations

Figure 4.6: Comparison between QPINN controller solution and known optimal controller

Chapter 5

Extensions

This chapter focuses on two topics. First, in Section 5.1, the selection of quadratic activation function parameters is explored with the proposal of two methods for selecting a, b, c to best approximate a ReLU (rectified linear unit) activation function. Second, Sections 5.2 and 5.3 present work related to universal approximation. One of the limitations of two-layer quadratic neural networks (QNNs) is that they do not have the universal approximation property of other two-layer networks [5, 6]. Instead, they represent a subset of quadratic forms of the given input in homogeneous coordinates (i.e., with a 1 appended to the end of the input vector). Sections 5.2 and 5.3 present work on understanding this limitation and potential avenues to overcome it. Section 5.2 delves into lifting and provides a theorem that shows that QNNs with monomial liftings have the universal approximation property. Section 5.3 explores the use of ensemble techniques to increase the representational capacity of QNNs without the need of lifting.

5.1 Approximation of ReLU Activation Function

In contrast to quadratic activations functions, ReLU (rectified linear unit) activation functions have seen widespread adoption in the neural network community. For this reason, this section will present the selection of quadratic parameters a, b, c to best approximate a

ReLU activation function in the sense of minimizing the mean squared error (MSE) and the maximum absolute error (MAE). The expression for a quadratic activation function is given as

$$\sigma_Q(z) = az^2 + bz + c \tag{5.1}$$

and, for a ReLU activation function, it is given as

$$\sigma_R(z) = \begin{cases} z & z \geq 0 \\ 0 & z < 0 \end{cases} \tag{5.2}$$

Let $\Delta_\sigma(z)$ be the difference between (5.1) and (5.2), defined as

$$\Delta_\sigma(z) \equiv \sigma_Q(z) - \sigma_R(z) = \begin{cases} az^2 + (b-1)z + c & z \geq 0 \\ az^2 + bz + c & z < 0 \end{cases} \tag{5.3}$$

The following two subsections will present different methodologies to select a, b, c to minimize either the MSE or MAE between the activation functions over an symmetric interval $z \in [-r, r], r > 0$.

5.1.1 Minimizing Mean Squared Error

The MSE between $\sigma_Q(z)$ and $\sigma_R(z)$ over the interval $z \in [-r, r], r > 0$ is given by

$$e_{mse} = \frac{1}{2r} \int_{-r}^{r} \Delta_\sigma^2(z) dz \tag{5.4}$$

Theorem 4. *Given $r > 0$, the solution of*

$$\begin{aligned} &\min_{a,b,c} e_{mse} \\ &s.t.\ (5.4), (5.3) \end{aligned} \tag{5.5}$$

is

$$a = \frac{15}{32r}, \quad b = \frac{1}{2}, \quad c = \frac{3r}{32} \tag{5.6}$$

Proof. Expanding the equation (5.4) by splitting the integral and using (5.3) yields

$$e_{mse} = \frac{1}{2r}\int_{-r}^{0}(az^2+bz+c)^2dz + \frac{1}{2r}\int_{0}^{r}(az^2+(b-1)z+c)^2dz \tag{5.7}$$

Taking the partial derivative of (5.7) with respect to a using the Leibniz integral rule yields

$$\begin{aligned}
\frac{\partial e_{mse}}{\partial a} &= \frac{1}{2r}\int_{-r}^{0}\frac{\partial}{\partial a}(az^2+bz+c)^2dz + \frac{1}{2r}\int_{0}^{r}\frac{\partial}{\partial a}(az^2+(b-1)z+c)^2dz \\
&= \frac{1}{2r}\int_{-r}^{0}2z^2(az^2+bz+c)dz + \frac{1}{2r}\int_{0}^{r}2z^2(az^2+(b-1)z+c)dz \\
&= \frac{1}{r}\int_{-r}^{0}(az^4+bz^3+cz^2)dz + \frac{1}{r}\int_{0}^{r}(az^4+(b-1)z^3+cz^2)dz \\
&= \frac{1}{r}\left[\frac{a}{5}z^5+\frac{b}{4}z^4+\frac{c}{3}z^3\right]_{-r}^{0} + \frac{1}{r}\left[\frac{a}{5}z^5+\frac{(b-1)}{4}z^4+\frac{c}{3}z^3\right]_{0}^{r} \\
&= \frac{2a}{5}r^4+\frac{2c}{3}r^2-\frac{1}{4}r^3
\end{aligned} \tag{5.8}$$

Taking the partial derivative of (5.7) with respect to b yields

$$\begin{aligned}
\frac{\partial e_{mse}}{\partial b} &= \frac{1}{2r}\int_{-r}^{0}\frac{\partial}{\partial b}(az^2+bz+c)^2dz + \frac{1}{2r}\int_{0}^{r}\frac{\partial}{\partial b}(az^2+(b-1)z+c)^2dz \\
&= \frac{1}{r}\left[\frac{a}{4}z^4+\frac{b}{3}z^3+\frac{c}{2}z^2\right]_{-r}^{0} + \frac{1}{r}\left[\frac{a}{4}z^4+\frac{(b-1)}{3}z^3+\frac{c}{2}z^2\right]_{0}^{r} \\
&= \frac{2b-1}{3}r^2
\end{aligned} \tag{5.9}$$

and with respect to c yields

$$\begin{aligned}
\frac{\partial e_{mse}}{\partial c} &= \frac{1}{2r}\int_{-r}^{0}\frac{\partial}{\partial c}(az^2+bz+c)^2dz + \frac{1}{2r}\int_{0}^{r}\frac{\partial}{\partial c}(az^2+(b-1)z+c)^2dz \\
&= \frac{1}{r}\left[\frac{a}{3}z^3+\frac{b}{2}z^2+cz\right]_{-r}^{0} + \frac{1}{r}\left[\frac{a}{3}z^3+\frac{(b-1)}{2}z^2+cz\right]_{0}^{r} \\
&= \frac{2a}{3}r^2-\frac{1}{2}r+2c
\end{aligned} \tag{5.10}$$

To minimize e_{mse}, the necessary conditions for optimality are thus

$$\frac{\partial e_{mse}}{\partial a} = 0 \iff \frac{2a}{5}r^4 + \frac{2c}{3}r^2 - \frac{1}{4}r^3 = 0 \qquad \iff \quad \frac{2a}{5}r^2 + \frac{2c}{3} - \frac{1}{4}r = 0 \tag{5.11a}$$

$$\frac{\partial e_{mse}}{\partial b} = 0 \iff \frac{2b-1}{3}r^2 = 0 \qquad \iff \quad b = \frac{1}{2} \tag{5.11b}$$

$$\frac{\partial e_{mse}}{\partial c} = 0 \iff \frac{2a}{3}r^2 - \frac{1}{2}r + 2c = 0 \qquad \iff \quad \frac{2a}{3}r^2 - \frac{1}{2}r + 2c = 0 \tag{5.11c}$$

Combining the conditions (5.11a) and (5.11c) yields

$$\frac{1}{3}2ar^2 - \frac{1}{2}r + 2c = \frac{1}{3}(\frac{5}{4}r - \frac{10c}{3}) - \frac{1}{2}r + 2c = 0 \iff c = \frac{3r}{32} \tag{5.12}$$

and since $r > 0$, substituting (5.12) into (5.11c) yields

$$\frac{1}{3}2ar^2 - \frac{1}{2}r + 2\left(\frac{3r}{32}\right) = 0 \iff a = \frac{15}{32r} \tag{5.13}$$

Taking the second derivative gives the Hessian matrix

$$\begin{bmatrix} \frac{\partial^2 e_{mse}}{\partial a^2} & \frac{\partial^2 e_{mse}}{\partial a \partial b} & \frac{\partial^2 e_{mse}}{\partial a \partial c} \\ \frac{\partial^2 e_{mse}}{\partial b \partial a} & \frac{\partial^2 e_{mse}}{\partial b^2} & \frac{\partial^2 e_{mse}}{\partial b \partial c} \\ \frac{\partial^2 e_{mse}}{\partial c \partial a} & \frac{\partial^2 e_{mse}}{\partial c \partial b} & \frac{\partial^2 e_{mse}}{\partial c^2} \end{bmatrix} = \begin{bmatrix} \frac{2}{5}r^4 & 0 & \frac{2}{3}r^2 \\ 0 & \frac{2}{3}r^2 & 0 \\ \frac{2}{3}r^2 & 0 & 2 \end{bmatrix} \tag{5.14}$$

Using Schur's complement,

$$2 > 0,$$

$$\begin{bmatrix} \frac{2}{5}r^4 & 0 \\ 0 & \frac{2}{3}r^2 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} \frac{2}{3}r^2 \\ 0 \end{bmatrix}\begin{bmatrix} \frac{2}{3}r^2 \\ 0 \end{bmatrix}^T = \begin{bmatrix} \frac{8}{45}r^4 & 0 \\ 0 & \frac{2}{3}r^2 \end{bmatrix} \succ 0 \tag{5.15}$$

since $r > 0$, thus (5.14) is positive definite. Therefore, e_{mse} is minimized by $a = \frac{15}{32r}, b = \frac{1}{2}$, and $c = \frac{3r}{32}$. □

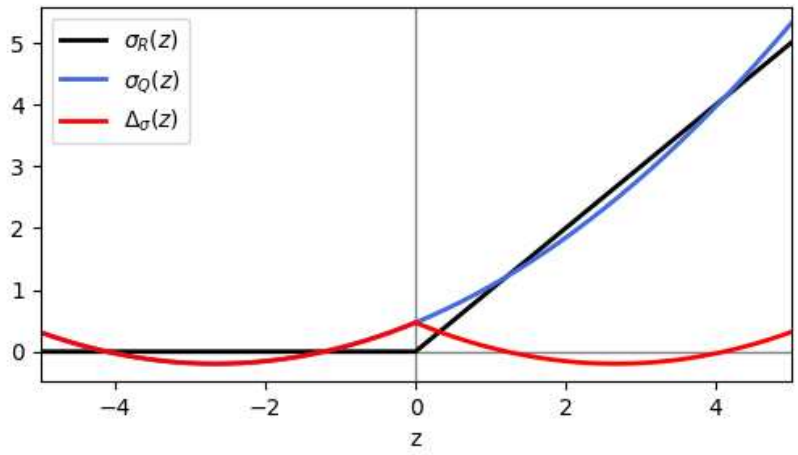


Figure 5.1: Comparison between ReLU (black) and quadratic activation function (blue) using $a = 0.0938$, $b = 0.5$, $c = 0.4688$, difference between the functions shown in red

Remark 10 (Constraints on a, b, c)**.** *Recall from Lemmas 2 and 7 that a requirement for the least squares training of QNNs and CQNNs is*

$$a > 0, \quad c > 0, \quad b^2 - 4ac \geq 0 \tag{5.16}$$

Using the a, b, c from (5.6) ensures that the conditions in (5.16) hold for any $r > 0$, i.e.,

$$r > 0 \implies a = \frac{15}{32r} > 0, \quad r > 0 \implies c = \frac{3r}{32} > 0$$
$$b^2 - 4ac = \left(\frac{1}{2}\right)^2 - 4\left(\frac{15}{32r}\right)\left(\frac{3r}{32}\right) = \frac{19}{256} \geq 0$$

Example 19 (Optimal a, b, c for $r = 5$)**.** *The optimal a, b, c which minimize (5.4) over the interval $[-5, 5]$ are*

$$a = 0.0938, \quad b = 0.5, \quad c = 0.4688$$

This result matches the proposed values of a, b, c from the authors of [12], who use least squares to fit the quadratic activation function to a ReLU activation function over the interval $[-5, 5]$. A comparison between $\sigma_R(z)$ and $\sigma_Q(z)$ using these parameter values is shown in Figure 5.1.

Remark 11 (Properties of Optimal a, b, c)**.** *There are two properties of note regarding the optimal a, b, c from (5.6). First, $a > 0, \forall r > 0$, meaning $\sigma_Q(z)$ always has positive curvature. This follows intuition, as $\sigma_Q(z)$ is trying to approximate $\sigma_R(z)$, which is a convex function. Second, with $b = \frac{1}{2}$, the error function $\Delta_\sigma(z)$ is even (i.e., $\Delta_\sigma(z) = \Delta_\sigma(-z)$). Furthermore, $b = \frac{1}{2}$ is a necessary condition for $\Delta_\sigma(z)$ to be even*

$$\Delta_\sigma(z) = az^2 + (b-1)z + c = a(-z)^2 + b(-z) + c = \Delta_\sigma(-z)$$
$$b - 1 = -b$$
$$b = \frac{1}{2}$$

It makes sense that the error function is even, because we are approximating over a symmetric interval $[-r, r]$, and thus neither positive nor negative z should result in more or less error. These two properties can be seen in an example shown in Figure 5.1.

5.1.2 Minimizing Maximum Absolute Error

The MAE between $\sigma_Q(z)$ and $\sigma_R(z)$ over an even interval $z \in [-r, r], r > 0$ is given by

$$e_{mae} = \max_{z \in [-r,r]} |\Delta_\sigma(z)| \tag{5.17}$$

For the problem of minimizing (5.17), we will consider $a > 0$ and $b = \frac{1}{2}$ following the rationale explained in Remark 11.

Theorem 5. *Given $r > 0$, a global solution of*

$$\begin{aligned} &\min_{a,b,c} e_{mae} \\ &s.t.\ (5.17), (5.3), a > 0, b = 0.5 \end{aligned} \tag{5.18}$$

is

$$a = \frac{1}{2r}, \quad b = \frac{1}{2}, \quad c = \frac{r}{16} \tag{5.19}$$

Moreover, when using (5.19),

$$|\Delta_\sigma(z)| \leq \frac{r}{16}, \forall z \in [-r, r] \tag{5.20}$$

Proof. In this proof $\vee$ and $\wedge$ represent the logical 'or' and 'and', respectively. Substituting $b = 0.5$ into (5.3) makes $\Delta_\sigma(z)$ an even function, which is given by

$$\begin{aligned} \Delta_\sigma(z) &= \begin{cases} az^2 - 0.5z + c & z \geq 0 \\ az^2 + 0.5z + c & z < 0 \end{cases} \\ &= az^2 - 0.5|z| + c \end{aligned} \tag{5.21}$$

Since $\Delta_\sigma(z)$ is even, the following holds

$$\max_{z\in[-r,r]} |\Delta_\sigma(z)| = \max_{z\in[-r,0]} |\Delta_\sigma(z)| = \max_{z\in[0,r]} |\Delta_\sigma(z)| \tag{5.22}$$

Additionally, notice that since $|\Delta_\sigma(z)| \geq 0, \forall z$ any maximizer of $\Delta_\sigma^2(z)$ will also be a maximizer of $|\Delta_\sigma(z)|$. Thus, in conjunction with (5.22), it can be seen that problem (5.18) has the same solution as

$$\begin{aligned} &\min_{a,b,c} \max_{z\in[0,r]} \Delta_\sigma^2(z) \\ &\text{s.t. } (5.3), a > 0, b = 0.5 \end{aligned} \tag{5.23}$$

The candidate maximizers of $\Delta_\sigma^2(z)$ from (5.23) are $z = 0$, $z = r$, and the critical points of $\Delta_\sigma^2(z)$ which lie in the interval $(0, r)$. The critical points are given by

$$\begin{aligned} &\frac{d(\Delta_\sigma^2(z))}{dz} = 2(az^2 - 0.5z + c)(2az - 0.5) = 0 \\ &\iff (az^2 - 0.5z + c) = 0 \vee (2az - 0.5) = 0 \end{aligned}$$

However, since $(az^2 - 0.5z + c) = 0$ implies $\Delta_\sigma^2(z) = 0$, its solutions are not maximizers.

Thus the candidates for the maxima are given by

$$\Delta_\sigma^2(0) = c^2 \tag{5.24a}$$

$$\Delta_\sigma^2\left(\frac{1}{4a}\right) = \left(c - \frac{1}{16a}\right)^2 \tag{5.24b}$$

$$\Delta_\sigma^2(r) = \left(ar^2 - \frac{1}{2}r + c\right)^2 \tag{5.24c}$$

Therefore,

$$\max_{z\in[0,r]} \Delta_\sigma^2(z) = \max\left[c^2, \left(c - \frac{1}{16a}\right)^2, \left(ar^2 - \frac{1}{2}r + c\right)^2\right] \tag{5.25}$$

from which it can be seen that $\max\limits_{z\in[0,r]} \Delta_\sigma^2(z) \geq c^2$, and thus the solution of the problem (5.23) is lower bounded by c^2, i.e.,

$$\min_{a,b,c} \max_{z\in[0,r]} \Delta_\sigma^2(z) \geq c^2$$

Therefore, a global solution of (5.23) is one which minimizes c^2, while also ensuring

$$\max_{z\in[0,r]} \Delta_\sigma^2(z) = c^2 \tag{5.26}$$

Thus, the final part of this proof has two steps. First, we find the required conditions for (5.26), then we minimize c^2 subject to those conditions. From (5.24a), (5.24b), and (5.24c) the condition for (5.26) are twofold. First, $\Delta_\sigma^2(\frac{1}{4a}) \leq \Delta_\sigma^2(0)$, which yields the condition

$$\Delta_\sigma^2\left(\frac{1}{4a}\right) \leq \Delta_\sigma^2(0) \iff \left(c - \frac{1}{16a}\right)^2 \leq c^2 \iff 0 \leq c^2 - \left(c - \frac{1}{16a}\right)^2$$

Recalling that $a > 0$, using the property $x^2 - y^2 = (x - y)(x + y), x \in \mathbb{R}, y \in \mathbb{R}$ yields

$$0 \leq c^2 - \left(c - \frac{1}{16a}\right)^2 \iff 0 \leq \left(\frac{1}{16a}\right)\left(2c - \frac{1}{16a}\right) \iff \frac{1}{32a} \leq c \tag{5.27}$$

Second, $\Delta_\sigma^2(r) \leq \Delta_\sigma^2(0)$, which yields the condition

$$\Delta_\sigma^2(r) \leq \Delta_\sigma^2(0) \iff \left(ar^2 - \frac{1}{2}r + c\right)^2 \leq c^2 \iff 0 \leq \left(-ar^2 + \frac{1}{2}r\right)\left(2c + ar^2 - \frac{1}{2}r\right) \tag{5.28}$$

Using (5.27)

$$\left(2c + ar^2 - \frac{1}{2}r\right) \geq \left(2\left(\frac{1}{32a}\right) + ar^2 - \frac{1}{2}r\right) = \frac{1}{16a}\left(4ar - 1\right)^2 \geq 0$$

Thus, the condition from (5.28) becomes

$$0 \leq \left(-ar^2 + \frac{1}{2}r\right)\left(2c + ar^2 - \frac{1}{2}r\right) \wedge 0 \leq \left(2c + ar^2 - \frac{1}{2}r\right) \implies a \leq \frac{1}{2r} \tag{5.29}$$

Therefore, using conditions (5.27) and (5.29), any solution of (5.30) is a solution of (5.23).

$$\begin{aligned} &\min_{a,b,c} c^2 \\ &\text{s.t. } \frac{1}{32c} \leq a \leq \frac{1}{2r}, a > 0, b = 0.5, r > 0 \end{aligned} \tag{5.30}$$

From $c \geq \frac{1}{32a}$, the minimum c is achieved by maximizing a subject to $a \leq \frac{1}{2}r$. This yields the optimal values $a = \frac{1}{2r}$ and $c = \frac{r}{16}$. Furthermore, from (5.25) it can be seen that using $a = \frac{1}{2r}$ and $c = \frac{r}{16}$ results in $\Delta_\sigma^2(z) \leq c^2 = \frac{r^2}{16^2}$, and thus $|\Delta_\sigma(z)| \leq c = \frac{r}{16}$ for all $z \in [-r, r]$. □

Remark 12 (Constraints on a, b, c)**.** *Recall from Lemmas 2 and 7 that a requirement for the least squares training of QNNs and CQNNs is*

$$a > 0, \quad c > 0, \quad b^2 - 4ac \geq 0 \tag{5.31}$$

Using the a, b, c from (5.19) ensures that the conditions in (5.31) hold for any $r > 0$, i.e.,

$$r > 0 \implies a = \frac{1}{2r} > 0, \quad r > 0 \implies c = \frac{r}{16} > 0$$
$$b^2 - 4ac = \left(\frac{1}{2}\right)^2 - 4\left(\frac{1}{2r}\right)\left(\frac{r}{16}\right) = \frac{1}{8} \geq 0$$

5.2 Lifting Quadratic Networks

This section explores the implications of lifting on the universal approximation property of two-layer QNNs. Recalling Section 2.2 of the Preliminaries, for an input vector $x \in \mathbb{R}^n$, the lifted input vector $z \in \mathbb{R}^{n+M}$ is given by

$$z = \begin{bmatrix} x^T & \Phi(x)^T \end{bmatrix}^T \tag{5.32}$$

where $\Phi(x) \in \mathbb{R}^M$ is the augmented input. For this section, monomial lifting will be used, for which $\Phi(x) = \Phi^{(d)}(x)$ contains all the monomials of degrees 2 to d formed from the elements of x when $d \geq 2$. For monomial lifting, the length of $\Phi^{(d)}(x)$ is given by

$$M = \sum_{k=2}^{d} \binom{n+k-1}{k} = \sum_{k=2}^{d} \left(\frac{(n+k-1)(n+k-2)\cdots(n)}{k(k-1)\cdots 1} \right) \tag{5.33}$$

In the case of $d = 1$, then $\Phi^{(d)}(x)$ is defined to be the empty vector.

Example 20 (Monomial lifting)**.** *Let $x = [v\ w]$, for $d = 3$, the augmented input $\Phi^{(3)}(x)$ and the lifted input z are given as*

$$\Phi^{(3)}(x) = \begin{bmatrix} v^2 & vw & w^2 & v^3 & v^2w & vw^2 & w^3 \end{bmatrix}^T$$
$$z = \begin{bmatrix} v & w & v^2 & vw & w^2 & v^3 & v^2w & vw^2 & w^3 \end{bmatrix}^T$$

The following theorem shows that using monomial lifting of a sufficiently high degree allows for two-layer QNNs to become universal approximators.

Theorem 6. *Let $x \in \mathbb{R}^n$ be an input vector and define z as the monomial lifted input*

$$z = \begin{bmatrix} x^T & (\Phi^{(d)}(x))^T \end{bmatrix}^T \tag{5.34}$$

where $\Phi^{(d)}(x) \in \mathbb{R}^M$ for $d \geq 2$ is a vector containing all monomials of degrees 2 to d formed

from the elements of x. In the case of $d = 1$, then $\Phi^{(d)}(x)$ is defined as the empty vector. If $\hat{y}(z)$ is the output of a QNN with $a > 0$, $c > 0$, and $b^2 - 4ac \geq 0$ using the lifted input (5.34), then for any function $y(x)$, which is continuous on a compact set C, then there exists a positive integer d such that $\|\hat{y}(z) - y(x)\|_2^2 < \epsilon$ for any $\epsilon > 0$ for all $x \in C$.

Proof. This proof is divided into two parts. First, we will show that a QNN with a monomial lifted input can represent any polynomial function of an arbitrary degree. Then, we will invoke the Stone-Weierstrass theorem to show universal approximation. Consider a real symmetric matrix Z of the form

$$Z = \begin{bmatrix} aZ_1 & \frac{b}{2}Z_2 \\ \frac{b}{2}(Z_2)^T & c\mathbf{Tr}(Z_1) \end{bmatrix}$$

For $a > 0$, $b \neq 0$, $c > 0$, invoking Lemma 5, there exists a decomposition $Z = Z^+ - Z^-$ where

$$Z^+ = \begin{bmatrix} aZ_1^+ & \frac{b}{2}Z_2^+ \\ \frac{b}{2}(Z_2^+)^T & c\mathbf{Tr}(Z_1^+) \end{bmatrix} \succeq 0, \quad Z^- = \begin{bmatrix} aZ_1^- & \frac{b}{2}Z_2^- \\ \frac{b}{2}(Z_2^-)^T & c\mathbf{Tr}(Z_1^-) \end{bmatrix} \succeq 0$$

For $a > 0$, $c > 0$, $b^2 - 4ac \geq 0$, from Lemma 6 it follows that

$$\begin{bmatrix} aZ_1^+ & \frac{b}{2}Z_2^+ \\ \frac{b}{2}(Z_2^+)^T & c\mathbf{Tr}(Z_1^+) \end{bmatrix} \succeq 0 \implies \begin{bmatrix} Z_1^+ & Z_2^+ \\ (Z_2^+)^T & \mathbf{Tr}(Z_1^+) \end{bmatrix} \succeq 0$$
$$\begin{bmatrix} aZ_1^- & \frac{b}{2}Z_2^- \\ \frac{b}{2}(Z_2^-)^T & c\mathbf{Tr}(Z_1^-) \end{bmatrix} \succeq 0 \implies \begin{bmatrix} Z_1^- & Z_2^- \\ (Z_2^-)^T & \mathbf{Tr}(Z_1^-) \end{bmatrix} \succeq 0$$

Therefore, any input-output expression of the form

$$\hat{y}(z) = \begin{bmatrix} z \\ 1 \end{bmatrix}^T \begin{bmatrix} aZ_1 & \frac{b}{2}Z_2 \\ \frac{b}{2}(Z_2)^T & c\mathbf{Tr}(Z_1) \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix} \tag{5.35}$$

can be equivalently written as

$$\hat{y}(z) = \begin{bmatrix} z \\ 1 \end{bmatrix}^T \begin{bmatrix} a(Z_1^+ - Z_1^-) & \frac{b}{2}(Z_2^+ - Z_2^-) \\ \frac{b}{2}(Z_2^+ - Z_2^-)^T & c\mathbf{Tr}(Z_1^+ - Z_1^-) \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix}$$
$$\begin{bmatrix} Z_1^+ & Z_2^+ \\ (Z_2^+)^T & \mathbf{Tr}(Z_1^+) \end{bmatrix} \succeq 0, \quad \begin{bmatrix} Z_1^- & Z_2^- \\ (Z_2^-)^T & \mathbf{Tr}(Z_1^-) \end{bmatrix} \succeq 0$$

which is the QNN expression from (2.8). In the special case that the desired function $y(x)$ can be written in the form (5.35) with z replaced by x, then by selecting $d = 1$, it follows that there exists a $\hat{y}(z)$ such that $\|\hat{y}(z) - y(x)\|_2^2 = 0$.

If this special case is not met, then let $d \geq 2$. If z contains all monomials of the components of x up to degree d, then it can be seen that $\hat{y}(z)$ is a polynomial of maximum degree $2d$, whose coefficients are given by the elements of Z_1 and Z_2, and who must satisfy the trace constraint from equation (5.35). For $x = [x_1 \; x_{2:n}^T]^T$, since $d \geq 2$, then z contains both x_1 and x_1^2. Without loss of generality, re-arrange z such that it is of the form

$$z = \begin{bmatrix} x_1 & x_1^2 & w^T \end{bmatrix}^T \in \mathbb{R}^{M+n}$$

where $w \in \mathbb{R}^q$ is a vector containing the remaining elements of z, with $q = M + n - 2$. Recalling that Z_1 is symmetric, the input-output expression is given by

$$\hat{y}(z) = \begin{bmatrix} x_1 \\ x_1^2 \\ w \\ 1 \end{bmatrix}^T \begin{bmatrix} aZ_{1_{\{1,1\}}} & aZ_{1_{\{1,2\}}} & aZ_{1_{\{1,3:q\}}} & \frac{b}{2}Z_{2_{\{1\}}} \\ aZ_{1_{\{1,2\}}} & aZ_{1_{\{2,2\}}} & aZ_{1_{\{2,3:q\}}} & \frac{b}{2}Z_{2_{\{2\}}} \\ a(Z_{1_{\{1,3:q\}}})^T & a(Z_{1_{\{2,3:q\}}})^T & aZ_{1_{\{3:q,3:q\}}} & \frac{b}{2}Z_{2_{\{3:q\}}} \\ \frac{b}{2}Z_{2_{\{1\}}} & \frac{b}{2}Z_{2_{\{2\}}} & \frac{b}{2}(Z_{2_{\{3:q\}}})^T & c\mathbf{Tr}(Z_1) \end{bmatrix} \begin{bmatrix} x_1 \\ x_1^2 \\ w \\ 1 \end{bmatrix}$$

Expanding the expression and collecting the like terms yields

$$\begin{aligned}\hat{y} = x_1^4(aZ_{1_{\{2,2\}}}) + x_1^3(2aZ_{1_{\{1,2\}}}) + x_1^2(aZ_{1_{\{1,1\}}} + bZ_{2_{\{2\}}}) + x_1(bZ_{2_{\{1\}}}) + (c\mathbf{Tr}(Z_1)) \\ + w^T(aZ_{1_{\{3:q,3:q\}}})w + w^T(bZ_{2_{\{3:q\}}}) + x_1^2(2aZ_{1_{\{2,3:q\}}})w + x_1(2aZ_{1_{\{1,3:q\}}})w\end{aligned} \tag{5.36}$$

where

$$\mathbf{Tr}(Z_1) = Z_{1_{\{1,1\}}} + Z_{1_{\{2,2\}}} + \mathbf{Tr}(Z_{1_{\{3:q,3:q\}}})$$

It can be seen that if $Z_{1_{\{1,1\}}} \leftarrow Z_{1_{\{1,1\}}} + a^{-1}\delta$ and $Z_{2_{\{2\}}} \leftarrow Z_{2_{\{2\}}} - b^{-1}\delta$, then only the coefficient of the constant term in equation (5.36) changes. Thus, if we want to make $\mathbf{Tr}(Z_1) = Z_4$ for arbitrary Z_4 then δ must be defined as $\delta = a(Z_4 - Z_{1_{\{1,1\}}} - Z_{1_{\{2,2\}}} - \mathbf{Tr}(Z_{1_{\{3:q,3:q\}}}))$. Therefore, when $d \geq 2$, $\hat{y}(z)$ can represent any polynomial, up to degree $2d$. Invoking the Stone-Weierstrass theorem [58], if $y(x)$ is continuous on a compact set C, then there exists a $d \geq 2$ such that $\|\hat{y}(z) - y(x)\|_2^2 < \epsilon$ for any $\epsilon > 0$ for all $x \in C$. □

5.3 Ensemble Quadratic Neural Networks

This section proposes ensemble techniques to increase the representational capacity of two-layer QNNs without the use of lifting. Ensemble techniques are a collection of methodologies which seek to increase the performance of learning algorithms by combining multiple learners or models. The premise is that by training multiple weak learners (i.e. models that may not be able to fully capture the relationship in the data) they can form an aggregated model which outperforms any of the individual base models.

5.3.1 Background

Although the concept predates their work, Brieman's bagging (bootstrap aggregating) [59] and Freund and Schapire's AdaBoost (adaptive boosting) [60] act as the basis for most modern ensemble techniques. Both algorithms use the premise of sampling the training set

to form multiple new training subsets, on which different base learners are trained. The base learners are then often combined using some form of averaging.

A prerequisite for successful bagging and boosting is high sensitivity of the base learners to changes in the training data (also called unstable learners in the literature) [59, 60]. By using sensitive base learners, the different training data tends to produce varied models, which aids in the aggregating process[1]. Due to the low-sensitivity of least squares as base learners [62], we speculate that least squares trained QNNs are also low-sensitivity learners. This claim will later be investigated in the examples provided in Section 5.3.4.1. To solve this limitation of bagging and boosting, Ting et al. [63, 64] proposed Feating (feature-subspace aggregating) as an ensemble technique which works with low-sensitivity models. The key idea behind feating is to train local models instead of global models. More specifically, feating partitions the feature space into multiple subspaces, then a local model is trained in each separate subspace. In contrast, global models ensemble techniques, such as bagging and boosting, pull samples from the entire feature space to train their models.

5.3.2 Bagging and Boosting

This section covers the bagging and boosting algorithms, as they will be used to show that QNNs are low-sensitivity learners in Section 5.3.4.1. Bagging [59] and Boosting [60] are two ensemble techniques that use training set sampling to generate multiple base learners in an iterative manner. At the start of training, the number of iterations B is selected. Each iteration trains a new base learner, thus B is also the total number of base learners in the ensemble. At each iteration, the training set inputs $X \in \mathbb{R}^{N \times n}$ and labels $Y \in \mathbb{R}^{N}$ are sampled (with replacement) to form $X_b \in \mathbb{R}^{N_b \times n}$ and $Y_b \in \mathbb{R}^{N_b}$ with $N_b = \lfloor \gamma N \rfloor$, where γ is the sampling ratio and $\lfloor \cdot \rfloor$ is the floor operation. A new base learner is then trained using X_b, Y_b. Bagging and boosting differ in the manner in which samples are selected during each iteration. During each bagging iteration, samples from X, Y are all equally likely to be

[1] A formal description of learner stability is provided in [61].

selected, whereas with boosting each sample is given a weight w_i representing its likelihood to be sampled. During the first iteration, all weights w_i are initialized equally. For subsequent iterations, each w_i is updated by evaluating the previous iteration's model on the training set X, Y. Samples with a higher error (or that were misclassified for classification problems) are then given larger weights. The standard bagging and boosting algorithms are described in Algorithms 1 and 2, respectively.

Algorithm 1: Bagging [59]

Input: X, Y, B, γ
Output: $\{\theta_b\}_{b=1}^{B}$
$X, Y \leftarrow$ Training set ;
$B \leftarrow$ Number of models to train ;
$\gamma \leftarrow$ Sampling ratio ;
$b \leftarrow 0$;
while $b < B$ **do**
    $X_b, Y_b \leftarrow \text{Sample}(X, Y, \gamma)$;
    $\theta_b \leftarrow \text{TrainNetwork}(X_b, Y_b)$;
    $b \leftarrow b + 1$;
end

Algorithm 2: Boosting [60]

Input: X, Y, B, γ
Output: $\{\theta_b\}_{b=1}^{B}$
$X, Y \leftarrow$ Training set ;
$B \leftarrow$ Number of models to train ;
$\gamma \leftarrow$ Sampling ratio ;
$w \leftarrow$ Initialize with equal weights ;
$b \leftarrow 0$;
while $b < B$ **do**
    $X_b, Y_b \leftarrow \text{Sample}(X, Y, w, \gamma)$;
    $\theta_b \leftarrow \text{TrainNetwork}(X_b, Y_b)$;
    $e_b \leftarrow \text{EvaluateError}(\theta_b, X, Y)$;
    $w \leftarrow \text{UpdateWeights}(w, e_b)$;
    $b \leftarrow b + 1$;
end

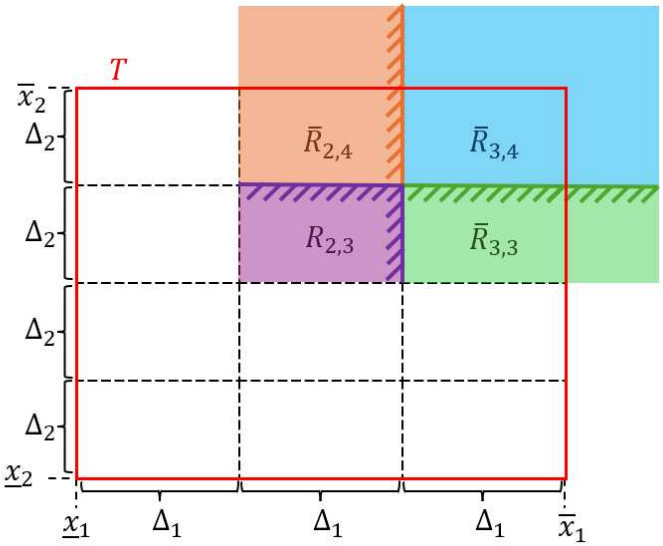


Figure 5.2: Example of input space division for $n = 2$, $r = \{3, 4\}$. Red square indicates training input space T, regions on edges extend to infinity denoted with $\bar{R}$, hatched lines indicate region ownership for boundaries

After the model is trained, bagging and boosting use majority voting for classification problems and averaging for regression problems when evaluating new data. However, with boosting each network's output is weighted based on its performance (i.e., loss function value) across the training set.

5.3.3 Local Model Ensemble Quadratic Neural Networks

This section proposes a methodology for the implementation of ensemble quadratic neural networks (EQNNs) based on feating ensembles [63, 64]. Since QNNs represent a subset of quadratic forms (see Section 3.2, Remark 5), by partitioning the feature space and training a QNN in each region, the input-output mapping of EQNNs become a piecewise quadratic function. It has been shown that piecewise affine functions are universal approximators [65, 66]. Therefore, piecewise quadratic functions, as a larger class of functions, are also universal approximators. Recently, this has been invoked by the authors of [67], who used piecewise quadratic functions to approximate arbitrary error functions. This motivates the investigation of EQNNs to increase the representational capability of QNNs.

Let $X \in \mathbb{R}^{N \times n}$ and $Y \in \mathbb{R}^N$ be the inputs and labels of a training set containing N samples, each of which has n features. Let $r = \{r_1, r_2, \ldots, r_n\}$ be the number of desired regions per feature. This implies that along the k^{th} feature there will be $r_k - 1$ breakpoints at which the model changes. Using the following definitions

$$\overline{x}_k = \max_{i=1,\ldots,N} x_{k_{\{i\}}}, \quad \underline{x}_k = \min_{i=1,\ldots,N} x_{k_{\{i\}}}, \quad \Delta_k = \frac{\overline{x}_k - \underline{x}_k}{r_k} \tag{5.37}$$

the training input space $T = \{x \in \mathbb{R}^n | x_k \in [\underline{x}_k, \overline{x}_k], \forall k = 1, \ldots, n\}$ is divided into $\prod_{k=1}^{n} r_k$ regions, each being a $\Delta_1 \times \Delta_2 \times \cdots \times \Delta_n$ hypercube. Each region is defined as

$$\begin{aligned} R_{i_1,i_2,\ldots,i_n} &= \{x \in T | x_k \in \bar{I}_{i_k,k}, \forall k = 1, \ldots, n\} \\ I_{1,k} &= [\underline{\mathrm{x}}_k, \underline{\mathrm{x}}_k + \Delta_k] \\ I_{i_k,k} &= \left(\underline{\mathrm{x}}_k + (i_k - 1)\Delta_k, \underline{\mathrm{x}}_k + (i_k)\Delta_k\right], \forall i_k \neq 1 \end{aligned} \tag{5.38}$$

where $i_k \in \{1, \ldots, r_k\}$ is the index for the k^{th} feature. In each region, a QNN is trained on the training samples which lie within the region. The EQNN training process is described in Algorithm 3. Using the EQNN to evaluate new data post-training, also called inference, the regions defined on the outer boundary of T are extended to infinity to capture data points that fall outside of T. The extended regions are defined as

$$\begin{aligned} \bar{R}_{i_1,i_2,\ldots,i_n} &= \{x \in T | x_k \in I_{i_k,k}, \forall k = 1, \ldots, n\} \\ \bar{I}_{1,k} &= (-\infty, \underline{x}_k + \Delta_k] \\ \bar{I}_{r_k,k} &= (\overline{x}_k - \Delta_k, \infty) \end{aligned} \tag{5.39}$$

An example is shown by regions $\bar{R}_{2,4}$, $\bar{R}_{3,4}$, and $\bar{R}_{3,3}$ in Figure 5.2. The inference process is described in Algorithm 4. From Lemma 1 (Chapter 2, Section 2.3), the upper bound on the number of hidden layer neurons required to implement the network is given by

$$m^* = 2(n+1) \prod_{k=1}^{n} r_k \tag{5.40}$$

Algorithm 3: EQNN Training

Input: r, X, Y, β, a, b, c
Output: EQNN, R, $\bar{R}$
$r \leftarrow$ List of desired regions per feature $\{r_1, \ldots, r_k\}$;
$X, Y \leftarrow$ Training data set of N samples with inputs having n features ;
$\beta \leftarrow$ Regularization coefficient ;
$a, b, c \leftarrow$ Quadratic activation function parameters ;
forall $k = \{1, \ldots, n\}$ **do**
  $\underline{\mathrm{x}}_k, \overline{x}_k, \Delta_k \leftarrow$ From (5.37) ;
  forall $i_k = \{1, \ldots, r_k\}$ **do**
    $I_{i_k,k} \leftarrow$ From (5.38) ;
  end
end
$R \leftarrow$ Generate all training regions using (5.38) and all $I_{i_k,k}$;
$\bar{R} \leftarrow$ Generate all inference regions using (5.39) and all $I_{i_k,k}$ on borders ;
foreach $R_{i_1,\ldots,i_k} \in R$ **do**
  $X_{i_1,\ldots,i_k}, Y_{i_1,\ldots,i_k} \leftarrow$ Samples from X, Y from region $R_{i_1,\ldots,i_k}$;
  $\mathrm{QNN}_{i_1,\ldots,i_k} \leftarrow$ Train LS QNN using (2.16) with β and a, b, c on $X_{i_1,\ldots,i_k}$, $Y_{i_1,\ldots,i_k}$;
end
$EQNN \leftarrow$ Store all trained QNNs with their associared regions ;

5.3.4 Examples

All quadratic networks use $a = 0.0937$, $b = 0.5$, $c = 0.4688$ for their activation function parameters following the optimal mean squared error ReLU approximation over the interval $[-5, 5]$, which is presented in Section 5.1.

5.3.4.1 Example 1: Bagging and Boosting of QNNs and CQNNs

This section provides some brief results that show the effect of bagging and boosting on QNNs and CQNNs. The results show that least squares trained QNNs and CQNNs are low-sensitivity learners.

Bagging and boosting were applied to CQNNs on the MNIST digit dataset [68] and to QNNs on the diabetes dataset [69], as both are well-known classification problems for the

Algorithm 4: EQNN Inference

Input: EQNN, R, $\bar{R}$, T x
Output: $\hat{y}$
$EQNN \leftarrow$ All QNNs alongwith associated regions ;
$R \leftarrow$ Set of all regions $R_{i_1,\ldots,i_k}$;
$\bar{R} \leftarrow$ Set of all regions $\bar{R}_{i_1,\ldots,i_k}$;
$T \leftarrow$ Training region ;
$x \leftarrow$ Input ;
if $x \in T$ **then**
  $R_x \leftarrow$ Get region from R to which x belongs ;
end
else
  $R_x \leftarrow$ Get region from $\bar{R}$ to which x belongs ;
end
$\text{QNN}_x \leftarrow$ Get QNN associated with region R_x ;
$\hat{y} \leftarrow \text{QNN}_x(x)$;

respective network architectures. The bagging and boosting procedures are given in Algorithms 1 and 2. For both examples and algorithms, $\gamma = 0.6$ was used. Furthermore, since bagging and boosting algorithms randomly generate their training subsets, the CQNN and QNN ensembles were retrained 10 and 100 times, respectively, and their results were averaged. Table 5.1 shows ensemble techniques for CQNNs applied to the MNIST digit dataset [68]. Table 5.2 shows ensemble techniques for QNNs applied to the diabetes classification dataset [69].

For both classification problems, bagging and boosting provided no benefit over the non-ensemble network. This example helps justify that least squared trained QNNs and CQNNs are low-sensitivity learners.

Table 5.1: MNIST classification testing accuracy and training time for various ensemble neural network solutions using $B = 10$, test accuracy averaged over 10 trials

	Test Accuracy [%]		
Network	Mean	Stand. Dev.	Mean Train Time [s]
CQNN	96.22	-	4.69
Bagging CQNN	95.78	1.15×10^{-3}	35.31
AdaBoost CQNN	95.79	4.91×10^{-4}	46.48

Table 5.2: Diabetes classification testing accuracy for various ensemble neural network solutions with different total number of trained networks, test accuracy averaged over 100 trials

	Test Accuracy [%]			
Network	$B = 1$	$B = 10$	$B = 50$	$B = 100$
QNN	74.03	-	-	-
Bagging QNN	-	73.03	73.90	73.92
AdaBoost QNN	-	73.89	74.09	74.03

5.3.4.2 Example 2: Violin String Dynamics

The position x and velocity v of a point on a violin string being bowed can be modeled as a friction-induced vibration (FIV) system of the form

$$\begin{aligned} \dot{x} &= v \\ m\dot{v} &= -kx + F(v_r) \end{aligned} \tag{5.41}$$

where k is the spring constant and $F(v_r)$ is the Stribeck friction given by

$$F(v_r) = \begin{cases} (v_r - 2)^2 & v_r \geq 0 \\ -(v_r + 2)^2 & v_r < 0 \end{cases} \tag{5.42}$$

where $v_r = v - v_{bow}$, with v_{bow} being the speed of the violin bow. For this example, the values of $k = 3$, $m = 3$, and $v_{bow} = 1$ were selected. An EQNN was trained to perform system identification using simulation data of this system where the inputs and outputs are constructed as follows

$$X_i = \begin{bmatrix} x_{i-1} & v_{i-1} \end{bmatrix}, \quad Y_i = \begin{bmatrix} x_i & v_i \end{bmatrix} \tag{5.43}$$

To generate training data, the system was discretized using forward Euler's method and simulated over the time interval $t \in [0, 5]$. A total of 60 simulated trajectories were obtained with randomized initial position $x_0 \in [-2, 5]$ and velocity $v_0 \in [-1, 3]$. The simulated

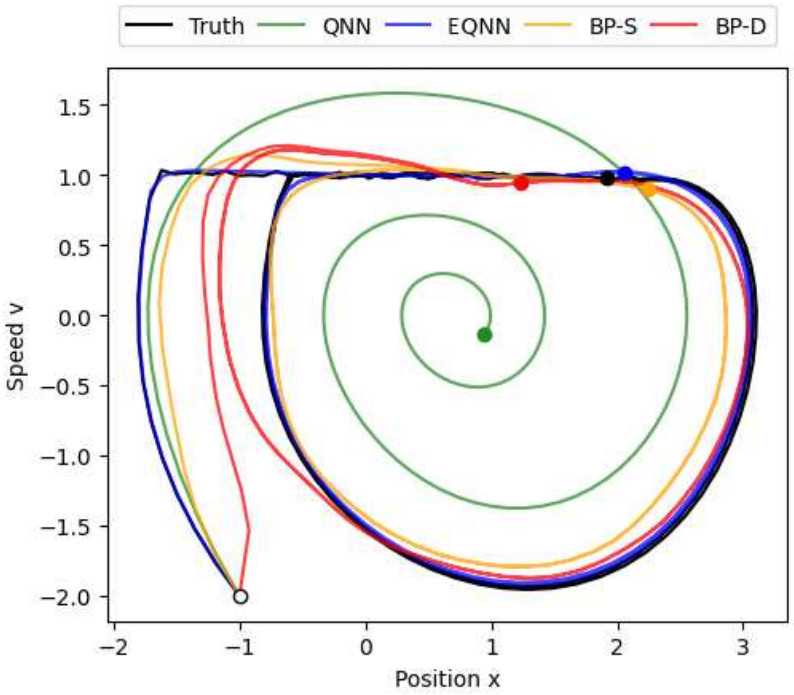


Figure 5.3: Comparison of state trajectories for auto-regressive predictions with $x_0 = -1$, $v_0 = -2$, initial state marked with a white circle, final states marked with a circle of corresponding color

trajectories were then sampled with a sampling period of 0.1 to generate the training data. In total, 3000 training samples were generated. Furthermore, the training data was subjected to additive Gaussian white noise $\nu \sim 0.01\mathcal{N}(0, 1)$ to mimic sensor noise.

To evaluate the performance of the EQNN composed of two-layer QNNs, it was compared to a least squares trained two-layer QNN [21], a two-layer backpropagation-trained neural network (BP-S), and a deep backpropagation-trained neural network (BP-D). To ensure the EQNN and backpropagation networks have a similar complexity level, the total number of hidden-layer neurons was fixed at 168. With this limit, the regions for the EQNN were selected as $r = \{2, 7\}$. The BP-S had a single hidden layer with 168 neurons, whereas the BP-D has 6 hidden layers, each with 28 neurons. Both the BP-S and BP-D were implemented using Python's PyTorch module and used the ReLU activation functions for their hidden layer neurons. Furthermore, the BP-S was trained over 1000 epochs, whereas the BP-D was trained over 1500 epochs, both using the stochastic gradient descent algorithm, with a

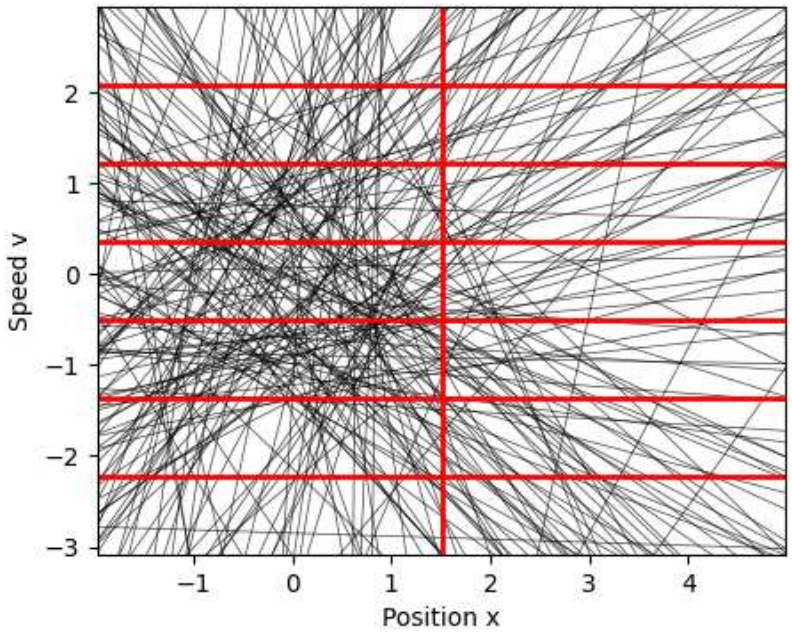


Figure 5.4: Comparison of piecewise model regions between EQNN and BP-S, black lines represent region boundaries for BP-S, red lines represent boundaries for EQNN

learning rate of 0.01, a momentum factor of 0.9, and a weight decay of 10^{-5}. Both the QNN and EQNN used $\beta = 10^{-5}$ for their least squares training. After training, the networks were then tested using their previous outputs as the inputs for the next prediction.

To compare the performance of each network, 100 simulations with initial conditions uniformly distributed over the ranges $x_0 \in [-2, 5]$, $v_0 \in [-1, 3]$ were performed. For each simulation, the mean squared error (MSE) between the predictions and the true states was recorded. The average MSE along with the training time of each network is shown in Table 5.3. Additionally, to show the differences in performance, an example simulation is shown in Figure 5.3. The EQNN's average MSE was the lowest of all the networks, showing that it was most accurate at modeling the dynamics of the system, along with a shorter training time than the backpropagation-trained networks. Furthermore, although one might expect the EQNN to have a larger training time compared to the QNN due to multiple networks being trained, the reduced number of data points per sub-network seems to offset this, which causes both the QNN and EQNN to have nearly identical training times. Moreover, it is

Table 5.3: Comparison of training time and average MSE for 100 auto-regressive trials with different initial conditions

Network	Average x MSE	Average v MSE	Training Time [s]
QNN	1.619	1.078	0.152
EQNN	2.99×10^{-2}	2.68×10^{-2}	0.150
BP-S	8.99×10^{-2}	7.90×10^{-2}	4.121
BP-D	4.97×10^{-1}	3.69×10^{-1}	13.443

important to note that the EQNN local networks were trained in series, and that parallel training may make EQNN training significantly faster compared to QNN training. In addition to the performance benefits, there are also improvements in explainability for EQNNs. Since both the ReLU network and the EQNN act as a form of piecewise model (ReLU being piecewise-affine and EQNN being piecewise-quadratic), Figure 5.4 compares the partitioning of the training input space between the models. Compared to the EQNN, the ReLU's partitioning is significantly more complex.

Chapter 6

Conclusions

Motivated by a growing need for explainable artificial intelligence, this thesis explores quadratic neural networks (QNNs) as a solution to many of the issues that limit the application of neural networks to safety-critical systems.

Chapters 3 and 4 show that, without regularization and mild constraints added on the activation function parameters and norm of the weights, the training of a two-layer convolutional quadratic neural network (CQNN) and a two-layer quadratic physics-informed neural network (QPINN) can be formulated as a least squares problem. In addition to the analytic expression for the globally optimal weights, the networks also have an input-output expression that is a quadratic form, which enables formal analysis in a system theory context. Furthermore, through examples related to system identification, sensor fusion, and solving boundary value problems, it is shown that CQNNs and QPINNs can achieve similar predictive performance compared to backpropagation-trained networks, whilst significantly reducing the training time and the number of hyperparameters that a designer would need to tune. Additionally, Chapter 5 presents the optimal quadratic activation function parameters to approximate a ReLU function by minimizing the mean squared error and the maximum absolute error. Table 6.1 presents an overview of backpropagation-trained networks and QNNs with respect to four key aspects.

Table 6.1: Comparison between backpropgation-trained networks and to-layer QNNs on four aspects which limit the application of neural networks to safety-critical systems

Aspect	BP-trained Network	Two-Layer QNNs
Explainability	Either do not provide an analytic input-output expression or provide one that is extremely complex to analyze	Quadratic form for input-output expression
Global Optimality	No guaratee of global optimal weights	Analytic solution for globally optimal weights (see Section 3.3 and Section 4.2.2)
Architecture & Hyperparameters	Heuristics for selection of number of neurons, layers, activation function, optimization algorithm and its parameters (e.g., learning rate, momentum)	Network architecture is a by-product of training, method for selection of quadratic activation function parameters presented in Section 5.1, regularization coefficient must be tuned
Training Time	Long training times and high computational demand due to iterative nature of backpropagation training	Short training times due to analytic epxression for globally optimal weights

This thesis also addresses one of the primary limitations of two-layer QNNs, that they do not possess the universal approximation property. In particular, this thesis proposes lifting and ensemble networks as a solution to increase the representational capacity of QNNs. In Section 5.2, it is proved that using monomial lifting of a sufficiently high degree allows QNNs to become universal approximators. Additionally, Section 5.3 provides an alternative to lifting by proposing ensemble QNNs (EQNNs), which train multiple QNNs by subdividing the feature space. EQNNs are shown to increase the representational capacity of QNNs in a system identification example.

6.1 Future Work

The following is a non-exhaustive list of potential future work directions:

- Deep QNNs with more than one hidden layer
- Generalization of CQNNs to use filters of higher dimensions

- Extension of the QPINNs methodology to handle nonlinear PDEs
- Systematic method for the selection of the least squares regularization term
- Exploration of optimal input-space partitioning for EQNNs

Bibliography

[1] B. Macukow, "Neural networks – state of art, brief history, basic models and architecture," in *Computer Information Systems and Industrial Management*, 2016, pp. 3–14.

[2] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, 2018.

[3] C. Thames and Y. Sun, "A survey of artificial intelligence approaches to safety and mission-critical systems," in *Integrated Communications, Navigation and Surveillance Conference (ICNS)*, Herndon, VA, USA, Apr. 2024, pp. 1–12.

[4] H. Forsberg, J. Lindén, J. Hjorth, T. Månefjord, and M. Daneshtalab, "Challenges in using neural networks in safety-critical applications," in *39th Digital Avionics Systems Conference (DASC)*, San Antonio, TX, USA, Nov. 2020, pp. 1–7.

[5] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.

[6] A. Pinkus, "Approximation theory of the MLP model in neural networks," *Acta Numerica*, vol. 8, pp. 143–195, 1999.

[7] T. Deng, "Effect of the number of hidden layer neurons on the accuracy of the back propagation neural network," *Highlights in Science, Engineering and Technology*, vol. 74, pp. 462–468, 12 2023.

[8] E. Delort, L. Riou, and A. Srivastava, "Environmental impact of artificial intelligence," INRIA, CEA Leti., Tech. Rep., Nov. 2023.

[9] "Artificial intelligence roadmap 2.0," EASA, Tech. Rep., May 2023.

[10] SAE, "G-34 artificial intelligence in aviation." [Online]. Available: https://standardsworks.sae.org/standards-committees/g-34-artificial-intelligence-aviation

[11] EUROCAE, "WG-114 artificial intelligence group." [Online]. Available: https://www.eurocae.net/working-group/wg-114/

[12] B. Bartan and M. Pilanci, "Neural spectrahedra and semidefinite lifts: Global convex optimization of polynomial activation neural networks in fully polynomial-time," 2021, arXiv:2101.02429.

[13] B. Bartan, "Distributed and error resilient convex optimization formulations in machine learning," Ph.D. dissertation, Stanford University, 2022.

[14] F. Fan, J. Xiong, and G. Wang, "Universal approximation with quadratic deep networks," *Neural Networks*, vol. 124, pp. 383–392, 2020.

[15] P. Mantini and S. K. Shah, "CQNN: Convolutional quadratic neural networks," in *2020 25th International Conference on Pattern Recognition (ICPR)*, Milan, Italy, Jan. 2020, pp. 9819–9826.

[16] F.-L. Fan, M. Li, F. Wang, R. Lai, and G. Wang, "On expressivity and trainability of quadratic networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 1, pp. 1228–1242, 2025.

[17] S. S. Du and J. D. Lee, "On the power of over-parametrization in neural networks with quadratic activation," in *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden, Dec. 2018.

[18] D. Gamarnik, E. C. Kizildağ, and I. Zadik, "Stationary points of shallow neural networks with quadratic activation function," 2020, arXiv:1912.01599.

[19] Y. Li, T. Ma, and H. Zhang, "Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations," in *Proceedings of the 31st Conference On Learning Theory*, July 2018, pp. 2–47.

[20] R. Livni, S. Shalev-Shwartz, and O. Shamir, "On the computational efficiency of training neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, Montreal, Canada, Dec. 2014, p. 855–863.

[21] L. Rodrigues, "Least squares solution for training of two-layer quadratic neural networks with applications to system identification," in *International Conference on Control, Decision and Information Technologies (CoDIT)*, Vallette, Malta, July 2024, pp. 916–921.

[22] L. Rodrigues and S. Givigi, "System identification and control using quadratic neural networks," *IEEE Control Systems Lett.*, vol. 7, pp. 2209–2214, 2023.

[23] E. H. Kerner, "Universal formats for nonlinear ordinary differential systems," *Journal of Mathematical Physics*, vol. 22, no. 7, pp. 1366–1371, July 1981.

[24] T. Ohtsuka, "Model structure simplification of nonlinear systems via immersion," *IEEE Transactions on Automatic Control*, vol. 50, no. 5, pp. 607–618, 2005.

[25] S. Klus and J.-P. N. N'konzi, "Data-driven system identification using quadratic embeddings of nonlinear dynamics," 2025, arXiv:2501.08202v2.

[26] F. Amato, C. Cosentino, and A. Merola, "On the region of asymptotic stability of nonlinear quadratic systems," in *14th Mediterranean Conference on Control and Automation*, Ancona, Italy, June 2006, pp. 1–5.

[27] C.-Y. Low, J. Park, and A. B.-J. Teoh, "Stacking-based deep neural network: Deep analytic network for pattern classification," *IEEE Transactions on Cybernetics*, vol. 50, no. 12, pp. 5021–5034, 2020.

[28] H. Zhuang, Z. Lin, Y. Yang, and K.-A. Toh, "An analytic formulation of convolutional neural network learning for pattern recognition," *Information Sciences*, vol. 686, 2025, Art. no. 121317.

[29] P. Guo and M. R. Lyu, "Pseudoinverse learning algorithm for feedforward neural networks," *Advances in Neural Networks and Applications*, pp. 321–326, 2001.

[30] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[31] H. Zhuang, Z. Lin, and K.-A. Toh, "Correlation projection for analytic learning of a classification network," *Neural Process. Lett.*, vol. 53, no. 6, p. 3893–3914, 2021.

[32] M. W. M. G. Dissanayake and N. Phan-Thien, "Neural-network-based approximations for solving partial differential equations," *Communications in Numerical Methods in Engineering*, vol. 10, no. 3, pp. 195–201, 1994.

[33] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, pp. 987–1000, 1998.

[34] M. Raissi, P. Perdikaris, and G. 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.

[35] S. Wang, S. Sankaran, H. Wang, and P. Perdikaris, "An expert's guide to training physics-informed neural networks," 2023, arXiv:2308.08468.

[36] R. Mattey and S. Ghosh, "A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations," *Computer Methods in Applied Mechanics and Engineering*, vol. 390, p. 114474, 2022.

[37] K. Haitsiukevich and A. Ilin, "Improved training of physics-informed neural networks with model ensembles," in *International Joint Conference on Neural Networks (IJCNN)*, Gold Coast, Australia, June 2023, pp. 1–8.

[38] T. Akhound-Sadegh, L. Perreault-Levasseur, J. Brandstetter, M. Welling, and S. Ravanbakhsh, "Lie point symmetry and physics-informed networks," in *37th Conference on Neural Information Processing Systems (NeurIPS)*, New Orleans, LA, USA, Dec. 2023.

[39] T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb, "Can physics-informed neural networks beat the finite element method?" *IMA Journal of Applied Mathematics*, vol. 89, no. 1, pp. 143–174, 2024.

[40] S. Desai, M. Mattheakis, H. Joy, P. Protopapas, and S. Roberts, "One-shot transfer learning of physics-informed neural networks," 2022, arXiv:2110.11286.

[41] R. Zhou, M. Fitzsimmons, Y. Meng, and J. Liu, "Physics-informed extreme learning machine Lyapunov functions," *IEEE Control Systems Letters*, vol. 8, pp. 1763–1768, 2024.

[42] J. Liu, M. Fitzsimmons, R. Zhou, and Y. Meng, "Formally verified physics-informed neural control Lyapunov functions," 2024, arXiv:2409.20528.

[43] T. Ohtsuka, "Model structure simplification of nonlinear systems via immersion," *IEEE Transactions on Automatic Control*, vol. 50, no. 5, pp. 607–618, 2005.

[44] C. Gu, "QLMOR: A projection-based nonlinear model order reduction approach using quadratic-linear representation of nonlinear systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 9, pp. 1307–1320, 2011.

[45] S. L. Brunton, M. Budišić, E. Kaiser, and J. N. Kutz, "Modern Koopman theory for dynamical systems," 2021, arXiv:2102.12086v2.

[46] "Essays and surveys in global optimization," 2005.

[47] Database for the identification of systems. [Online]. Available: https://homes.esat.kuleuven.be/~smc/daisy/index.html

[48] Y. Yao, X. Xu, C. Zhu, and C.-Y. Chan, "A hybrid fusion algorithm for GPS/INS integration during GPS outages," *Measurement*, vol. 103, pp. 42–51, 2017.

[49] Y. Liu, Q. Luo, and Y. Zhou, "Deep learning-enabled fusion to bridge GPS outages for INS/GPS integrated navigation," *IEEE Sensors J.*, vol. 22, no. 9, pp. 8974–8985, 2022.

[50] S. Taghizadeh and R. Safabakhsh, "An integrated INS/GNSS system with an attention-based deep network for drones in GNSS denied environments," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 38, no. 8, pp. 14–25, 2023.

[51] R. Gonzalez, C. Catania, P. Dabove, C. Taffernaberry, and M. Piras, "Model validation of an open-source framework for post-processing INS/GNSS systems," in *Proceedings of the 3rd International Conference on Geographical Information Systems Theory, Applications and Management (GISTAM)*, Porto, Portugal, Apr. 2017, pp. 201–208.

[52] R. Gonzalez, J. Giribet, and H. Patiño, "Navego: A simulation framework for low-cost integrated navigation systems," *Control Engineering and Applied Informatics*, vol. 17, no. 2, pp. 110–120, 2015.

[53] P. J. Olver, *Applications of Lie Groups to Differential Equations*. New York: Springer, 1993.

[54] A. Cohen, *An Introduction to The Lie Theory of One-Parameter Groups*. New York: C.C. Health & Co., 1911.

[55] A. Botros, K. Walker, and L. Rodrigues, "A toolbox for optimal control based on symmetry analysis with applications to max endurance and max range," 2018.

[56] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," 2017.

[57] ——, "Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations," 2017, arXiv:1711.10566.

[58] M. H. Stone, "The generalized Weierstrass approximation theorem," *Mathematics Magazine*, vol. 21, no. 5, pp. 237–254, 1948.

[59] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[60] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *13th International Conference on Machine Learning*, Bari, Italy, July 1996, p. 148–156.

[61] O. Bousquet and A. Elisseeff, "Stability and generalization," *Journal of Machine Learning Research*, vol. 2, pp. 499–526, 06 2002.

[62] A. Cohen, M. Davenport, and D. Leviatan, "On the stability and accuracy of least squares approximations," *Foundations of Computational Mathematics*, vol. 13, pp. 819–834, Mar. 2013.

[63] K. M. Ting, J. R. Wells, S. C. Tan, S. W. Teng, and G. I. Webb, "Feature-subspace aggregating: ensembles for stable and unstable learners," *Machine Learning*, vol. 82, no. 2, pp. 375–397, 2011.

[64] K. M. Ting, L. Zhu, and J. R. Wells, "Local models - the key to boosting stable learners successfully," *Computational Intelligence*, vol. 29, no. 2, pp. 331–356, 2012.

[65] P. Julian, A. Desages, and O. Agamennoni, "High-level canonical piecewise linear representation using a simplicial partition," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 46, no. 4, pp. 463–480, 1999.

[66] M. Johansson, "Piecewise linear control systems," Ph.D. dissertation, Lund University, 1999.

[67] A. Z. A. N. Gorban, E. M. Mirkes, "Piece-wise quadratic approximations of arbitrary error functions for fast and robust machine learning," 2016, arXiv:1605.06276.

[68] MNIST dataset. [Online]. Available: https://www.kaggle.com/datasets/hojjatk/mnist-dataset

[69] Diabetes dataset. [Online]. Available: https://www.kaggle.com/datasets/mathchi/diabetes-data-set