August 2014

Newton's Method Backpropagation for Complex-Valued Holomorphic Neural Networks: Algebraic and Analytic Properties

Diana Thomson La Corte
University of Wisconsin-Milwaukee

Newton's Method Backpropagation for Complex-Valued Holomorphic Neural Networks: Algebraic and Analytic Properties

by

Diana Thomson La Corte

A Dissertation Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

in

Mathematics

at

The University of Wisconsin-Milwaukee

August 2014

ABSTRACT

Newton's Method Backpropagation for Complex-Valued Holomorphic Neural Networks: Algebraic and Analytic Properties

by

Diana Thomson La Corte

The University of Wisconsin-Milwaukee, 2014

Under the Supervision of Professor Yi Ming Zou

The study of Newton's method in complex-valued neural networks (CVNNs) faces many difficulties. In this dissertation, we derive Newton's method backpropagation algorithms for complex-valued holomorphic multilayer perceptrons (MLPs), and we investigate the convergence of the one-step Newton steplength algorithm for the minimization of real-valued complex functions via Newton's method. The problem of singular Hessian matrices provides an obstacle to the use of Newton's method backpropagation to train CVNNs. We approach this problem by developing an adaptive underrelaxation factor algorithm that avoids singularity of the Hessian matrices for the minimization of real-valued complex polynomial functions.

To provide experimental support for the use of our algorithms, we perform a comparison of using sigmoidal functions versus their Taylor polynomial approximations as activation functions by using the Newton and pseudo-Newton backpropagation algorithms developed here and the known gradient descent backpropagation algorithm. Our experiments indicate that the Newton's method based algorithms, combined with the use of polynomial activation functions, provide significant improvement in the number of training iterations required over the existing algorithms. We also test

our underrelaxation factor algorithm using a small-scale polynomial neuron and a polynomial MLP. Finally, we investigate the application of an algebraic root-finding technique to the case of a polynomial MLP to develop a theoretical framework for the location of initial weight vectors that will guarantee successful training.

To Mom and Dad

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

I would first like to thank my advisor, Dr. Yi Ming Zou, for his guidance and support throughout the process of writing this dissertation. His insights have been inspiring and sparked many creative ideas, and his feedback on my work has always been timely and thorough–two traits of an advisor that are invaluable to a graduate student! I would also like to thank the rest of my dissertation committee, Drs. Allen Bell, Jeb Willenbring, Gabriella Pinter, and Peter Hinow, for their support during my graduate years, as well as Dr. Ian Musson for his contribution to my education in algebra. Special thanks go to Dr. Willenbring, who introduced me to the exciting world of algebra my freshman year in college and inspired me to specialize in this subject, and Dr. Craig Guilbault, who welcomed me both as an undergraduate and as a graduate student to UW-Milwaukee.

I must also acknowledge the incredible support of my family during the many years I have been a student. To my parents Barbara and Jim, whose love and encouragement have cheered me through the difficult periods of college and graduate school alike. To my grandmother Liz, who has been so generous in order to help me through this program. To my husband Jason, who has suffered through the perils of late nights and looming deadlines by my side. And to Djehuty, who has been my constant companion since I first started on this project, and who did his best to try and distract me whenever he could. Now we should have a lot more time to play ball!

Chapter 1

Introduction

Artificial neural networks (ANNs) are mathematical models composed of interconnected processing units meant to mimic biological neural networks. ANNs are distinguished from traditional computer progams by their ability to learn by example. Presented with a data set, an ANN creates an internal representation that it can then use to process new data. The use of fully complex-valued neural networks (CVNNs) to solve real-valued as well as complex-valued problems in physical applications has become increasingly popular in the neural network community in recent years [24, 36]. CVNNs suit a diverse variety of applications in mathematics, engineering, and the sciences, including function approximation, classification problems, image compression, speech recognition, and signal processing [23]. It is thus a worthwhile enterprise to study the training algorithms for CVNNs, as improvement here can result in greater efficiency in applications.

Complex-valued neural networks pose unique problems as opposed to their real-valued counterparts. Consider the problem of choosing the activation functions for a neural network. Real-valued activation functions for real-valued neural networks (RVNNs) are commonly taken to be everywhere differentiable and bounded. Typical activation functions used for RVNNs are the sigmoidal, hyperbolic tangent, and hyperbolic secant functions

$$f(x) = \frac{1}{1+\exp(-x)}, \ \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \text{ and } \operatorname{sech}(x) = \frac{2}{e^x + e^{-x}}.$$

For activation functions of CVNNs, an obvious choice is to use the complex coun-

terparts of these real-valued functions. However, as complex-valued functions, these functions are no longer differentiable and bounded near 0, since they have poles near 0. Different approaches have been proposed in the literature to address this problem.

Liouville's Theorem tells us that there is no non-constant complex-valued function which is both bounded and differentiable on the whole complex plane [11]. On the basis of Liouville's Theorem, [17] asserts that an entire function is not suitable as an activation function for a CVNN and claims boundedness as an essential property of the activation function. Some authors followed this same reasoning and use the so-called "split" functions of the type $f(z) = f(x + iy) = f_1(x) + if_2(y)$ where f_1, f_2 are real-valued functions, typically taken to be one of the sigmoidal functions [26, 27, 39]. Such activation functions have the advantage of easily modeling data with symmetry about the real and imaginary axes. However, this yields complex-valued neural networks which are close to real-valued networks of double dimensions and are not fully complex-valued [24]. Amplitude-phase-type activation functions have the type $f(z) = f_3(|z|)\exp(i\arg(z))$ where f_3 is a real-valued function. These process wave information well, but have the disadvantage of preserving phase data, making the training of a network more difficult [24, 27].

Some authors forgo complex-valued activation functions entirely, choosing instead to scale the complex inputs using bounded real-valued functions which are differentiable with respect to the real and imaginary parts [2–4]. While this approach allows for more natural grouping of data for classification problems, it requires a modified backpropagation algorithm to train the network, and again the networks are not fully complex-valued. Other authors choose differentiability over boundedness and use elementary transcendental functions [21, 27, 28]. Such functions have been used in CVNNs trained using the traditional gradient descent backpropagation algorithm and in other applications [10, 34, 46]. However, the problem of the existence of poles in a bounded region near 0 presents again. Though one can try to scale the data to avoid the regions which contain poles [32], this does not solve the problem, since for unknown composite functions, the locations of poles are not known *a priori*. The exponential function $\exp(z)$ has been proposed as an alternative to the elementary transcendental functions for some CVNNs, and experimental evidence suggests better performance of the entire exponential function as activation function than those with poles [42].

In this dissertation, we will derive the backpropagation algorithm for CVNNs based on Newton's method. We compare the performances of using the complex-valued sigmoidal activation function and its Taylor polynomial approximations. Our results give strong supporting evidence for the use of holomorphic functions as activation functions for CVNNs. Holomorphic functions encompass a general class of functions that are commonly used as activation functions. They allow a wide variety of choices both for activation functions and training methods. The differentiability of holomorphic functions leads to much simpler formulas in the backpropagation algorithms.

We pay particular attention to the use of polynomial functions as activation functions for CVNNs. Polynomials have been used in fully complex-valued functional link networks [5, 6], however their use is limited as activation functions for fully complex-valued networks. Polynomial functions are differentiable on the entire complex plane and are underlying our computations due to Taylor's Theorem, and they are bounded over any bounded region. Moreover, the complex Stone-Weierstrass Theorem implies that any continuous complex-valued function on a compact subset of the complex plane can be approximated by a polynomial [43]. Due to the nature of the problems associated with the activation functions in CVNNs, different choices of activation functions can only suit different types of neural networks properly, and one should only expect an approach to be better than the others in certain applications.

The use of polynomial functions as activation functions opens up possibilities otherwise not available, such as the theoretical study of using Newton's method in network algorithms. Newton's method is known to provide faster convergence than the commonly used gradient descent method whenever applicable. However, the study of using Newton's method in neural networks faces difficulties due to the complexity of the activation functions used, especially for the complex-valued networks. The split functions are out of consideration, since they are not differentiable. For other complex-valued direct extensions of the real-valued activation functions, in addition to the problem that the corresponding Hessian could be singular, which is faced by their real counterparts, the entries of the Hessian can also hit the poles. In this dissertation, we address two problems that arise from the study of Newton's method in the backpropagation algorithm for CVNNs with polynomial activation functions: the problem of singularity of the Hessian matrices, and the sensitivity of

Newton's method to the initial choice of iterates. In both cases, the polynomial activation functions allow us to take an algebraic approach to the problems. Another reason for studying CVNNs using polynomials as activation functions is that Newton's method is closely related to the field of complex dynamical systems [19, 25]. This connection has yet to obtain sufficient attention in the complex-valued neural network community, however this will not be developed here.

This dissertation is organized in the following manner.

In Chapter 2, we describe the basic setup of the artificial neuron and the complex-valued multilayer perceptron (MLP) and introduce the notation we will use throughout the remainder of the dissertation. We then describe the standard gradient descent backpropagation algorithm as it pertains to our setting.

In Chapter 3, we derive a recursive algorithm given by Theorem 3.2.1 to compute the entries of the Hessian matrices in the application of Newton's method to the backpropagation algorithm for complex-valued holomorphic MLPs. Corollary 3.3.1 gives a recursive algorithm for the application of the pseudo-Newton method to the backpropagation algorithm based on Theorem 3.2.1. We then develop the one-step Newton steplength algorithm for the minimization of real-valued complex functions using Newton's method which is given by Theorem 3.4.4, and compare our algorithms to the standard gradient-descent algorithm for a complex-valued MLP trained with data from the XOR problem to show the superiority of our methods.

In Chapter 4, we address the problem of singular Hessian matrices in training a complex-valued polynomial MLP using Newton's method. In Theorem 4.1.1, we give an adaptive underrelaxation factor algorithm that guarantees nonsingularity of the Hessian matrices for minimization of real-valued complex polynomial functions using Newton's method. Corollary 4.1.2 gives a similar result for the pseudo-Newton method based on Theorem 4.1.1. We test our adaptive underrelaxation factor algorithm for the pseudo-Newton case on both an artificial neuron and also on the XOR problem.

In Chapter 5, we investigate the application of the algebraic root-finding algorithm given in [40] to the case of complex-valued polynomial neurons and polynomial MLPs, and we develop a theoretical framework and search strategy based on the training set for the location of initial weight vectors that will guarantee successful training. A search algorithm for the complex-valued polynomial neuron is given in Proposition 5.3.3.

Chapter 2

Complex-Valued Neural Networks

There are a great variety of different types of artificial neural networks available for use in different applications. Our focus is on improving training processes for a particular type of artificial neural network called a multilayer perceptron (MLP). In this chapter, we develop the basic definitions and notation that we will use throughout the rest of this dissertation. We describe the setup of an artificial neuron and discuss the network architecture of complex-valued MLPs. The typical training algorithm used for complex-valued neural networks is the gradient descent backpropagation algorithm. We reformulate this standard training algorithm to our setting of holomorphic MLPs in order to gain perspective for the application of Newton's method to the backpropagation algorithm in Chapter 3.

2.1 Artificial Neurons: The Building Blocks of Neural Networks

The basic unit of any artificial neural network is the artificial neuron. Mathematically, an artificial neuron is a multivariate function meant to mimic a biological neuron. Let D_1 and D_2 be domains. Typically, $D_1 = D_2 = \mathbb{R}$ or $\mathbb{C}$, yielding a real- or complex-valued neuron, respectively. Computation in a generic neuron as shown in Figure 2.1 is given as follows. The **input** of the neuron is the vector $(x_1, ..., x_n) \in D_1^n$. The variables $w_1, ..., w_n \in D_1$ are called the **weights** of the

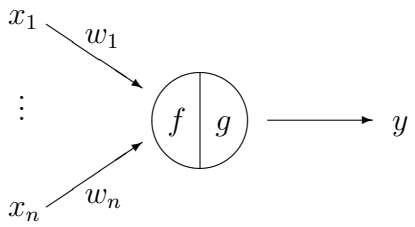


Figure 2.1: An Artificial Neuron

neuron. The **output** of the neuron is given by

$$y = g\left(f(w_1, ..., w_n; x_1, ..., x_n)\right) \in D_2,$$

where $f : D_1^n \to D_1$ is a function of the weights called the **propagation function** of the neuron, and $g : D_1 \to D_2$ is called the **activation function** of the neuron. The propagation function typically computes a linear combination of the input vectors with the weight vectors, sometimes with an added bias term $\theta \in D_1$ [24]:

$$y = g\left(\sum_{i=1}^{n} w_i x_u + \theta\right).$$

In our setting, we look at complex neurons with $D_1 = D_2 = \mathbb{C}$ and we use a zero bias term $\theta = 0$. The neuron is trained using a training set with N data points $\{(z_{t1}, ..., z_{tm}, d_t) \,|\, t = 1, ..., N\}$ by minimizing the sum-of-squares error function

$$E = \sum_{t=1}^{N} |y_t - d_t|^2 = \sum_{t=1}^{N} (y_t - d_t)(\overline{y_t} - \overline{d_t}).$$

2.2 Holomorphic MLPs: Definition and Network Architecture

A well-used type of artificial neural network is the multilayer perceptron (MLP). An MLP is built of several layers of single neurons hooked together by a network of weight vectors. Usually the activation function is taken to be the same among a single layer of the network; the defining characteristic of the MLP is that in at least one layer, the activation function must be nonlinear. If there is no nonlinear activation function, the network can be collapsed to a two-layer network [9].

Definition 2.2.1. A **holomorphic MLP** is a complex-valued MLP in which the activation function in the layer indexed by p of the network is holomorphic on some domain $\Omega_p \subseteq \mathbb{C}$.

Most of the publications on complex-valued neural networks with holomorphic activation functions deal with functions that have poles. We will mainly focus on entire functions for the purpose of applying Newton's method. For these functions, we do not have to worry about the entries of a Hessian matrix hitting the poles, however the matrix itself can be singular. However, we will allow some flexibility in our setting and set up our notation for a general L-layer holomorphic MLP as follows (see Figure 2.2).

- The input layer has $m = K_0$ input nodes denoted

$$z_1 = x_1^{(0)}, ..., z_m = x_m^{(0)}.$$

- There are $L-1$ hidden layers of neurons, and the pth ($1 \leq p \leq L-1$) hidden layer contains K_p nodes. We denote the output of node j ($j = 1, ..., K_p$) in the pth layer by $x_j^{(p)}$. The inputted weights to the pth layer are denoted $w_{ji}^{(p-1)}$ ($j = 1, ..., K_p$, $i = 1, ..., K_{p-1}$), where j denotes the target node of the weight in the pth layer and i denotes the source node in the $(p-1)$th layer. With these conventions we define the weighted net sum and the output of node j of the pth layer by

$$\left(x_j^{(p)}\right)^{\text{net}} = \sum_{i=1}^{K_{p-1}} w_{ji}^{(p-1)} x_i^{(p-1)} \text{ and } x_j^{(p)} = g_p\left(\left(x_j^{(p)}\right)^{\text{net}}\right),$$

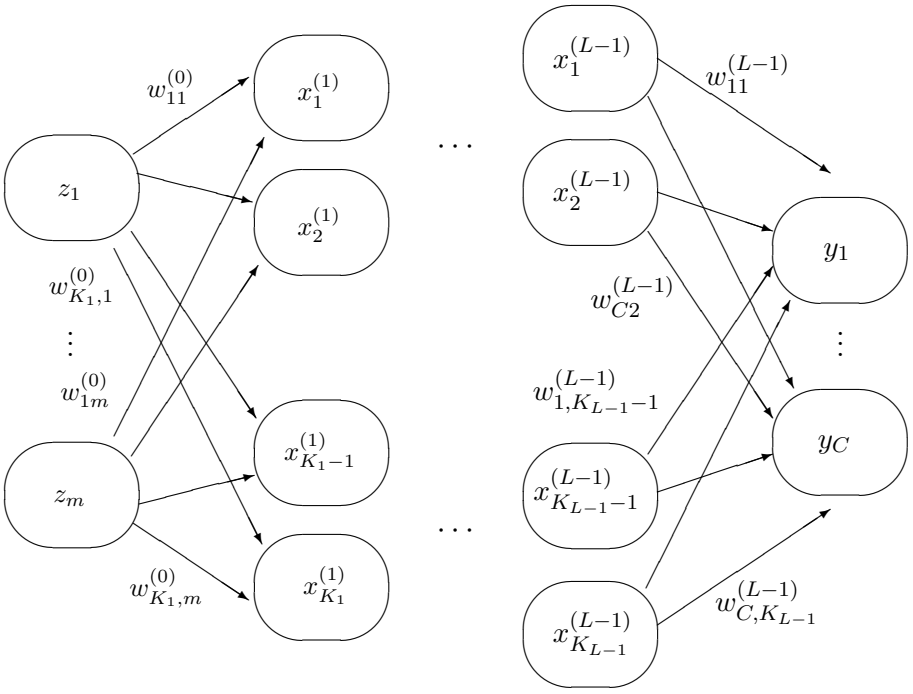


Figure 2.2: Network Architecture

where g_p, which is assumed to be holomorphic on some domain $\Omega_p \subseteq \mathbb{C}$, is the activation function for all the neurons in the pth layer.

- The output layer has $C = K_L$ output nodes denoted $y_1 = x_1^{(L)}, ..., y_C = x_C^{(L)}$. We define the weighted net sum and the output of node l ($l = 1, ..., C$) by

$$y_l^{\text{net}} = \sum_{k=1}^{K_{L-1}} w_{lk}^{(L-1)} x_k^{(L-1)} \text{ and } y_l = g_L\left(y_l^{\text{net}}\right),$$

where g_L, which is assumed to be holomorphic on some domain $\Omega_L \subseteq \mathbb{C}$, is the activation function for all the neurons in the output layer.

To train the network, we use a training set with N data points

$$\{(z_{t1}, ..., z_{tm}, d_{t1}, ..., d_{tC}) \,|\, t = 1, ..., N\},$$

where $(z_{t1}, ..., z_{tm})$ is the input vector corresponding to the desired output vector $(d_{t1}, ..., d_{tC})$. As the input vector $(z_{t1}, ..., z_{tm})$ of the tth training point is propagated

throughout the network we update the subscripts of the network calculations with an additional t subscript to signify that those values correspond to the tth training point. For example, $(x_{tj}^{(p)})^{\text{net}}$, $x_{tj}^{(p)}$, y_{tl}^{net}, and y_{tl}. Finally, we train the network by minimizing the standard sum-of-squares error function

$$E = \frac{1}{N}\sum_{t=1}^{N}\sum_{l=1}^{C}|y_{tl} - d_{tl}|^2 = \frac{1}{N}\sum_{t=1}^{N}\sum_{l=1}^{C}(y_{tl} - d_{tl})\left(\overline{y_{tl}} - \overline{d_{tl}}\right). \tag{2.2.1}$$

2.3 The Gradient Descent Backpropagation Algorithm

Minimization of the error function can be achieved through the use of the backpropagation algorithm. Backpropagation trains the network by updating the output layer weights first in each step (via an update rule from some numerical minimization algorithm), then using the updated output layer weights to update the first hidden layer weights, and so on, "backpropagating" the updates throughout the network until a desired level of accuracy is achieved (usually, this is when the error function drops below a pre-fixed value). In the case of real-valued neural networks, minimization of the error function by Newton's method is generally thought to be too computationally "expensive," and several different methods are commonly used to approximate the Hessian matrices instead of computing them directly: for example the conjugate gradient, truncated Newton, Gauss-Newton and Levenberg-Marquardt algorithms [1, 8, 20, 36, 48]. In contrast, for complex-valued neural networks, gradient descent methods, which are known to give stable (albeit slow) convergence, are commonly used due to their relatively simple formulations, and a number of such minimization algorithms exist [18, 32, 49].

We reformulate a backpropagation algorithm using gradient descent according to our setting of the neural networks defined in Section II for two reasons: the algorithm has a much simpler formulation compared with the known ones [9, 32] due to the activation functions being taken to be holomorphic, and we will use it for comparison purpose. A similar formulation of the backpropagation algorithm to ours is presented in [33]. The formulas of gradient descent for complex functions can be found in [29]. We use the following vector notation. For $1 \leq p \leq L$, we

denote the weights that input into the pth layer of the network using a vector whose components correspond to the target nodes:

$$\mathbf{w}^{(p-1)} := \left(w_{11}^{(p-1)}, ..., w_{1K_{p-1}}^{(p-1)}, ..., w_{K_p1}^{(p-1)}, ..., w_{K_pK_{p-1}}^{(p-1)}\right)^T,$$

that is, the components of $\mathbf{w}^{(p-1)}$ are

$$\mathbf{w}^{(p-1)}\left[(j-1)\cdot K_{p-1}+i\right] = w_{ji}^{(p-1)}, \tag{2.3.1}$$

where $j = 1, ..., K_p$, $i = 1, ..., K_{p-1}$. Using this notation the update steps for back-propagation look like

$$\mathbf{w}^{(p-1)}(n+1) = \mathbf{w}^{(p-1)}(n) + \mu(n)\Delta\mathbf{w}^{(p-1)}, \tag{2.3.2}$$

where $\mathbf{w}^{(p-1)}(n)$ denotes the weight value after the nth iteration of the training algorithm, and $\mu(n)$ denotes the learning rate or steplength which is allowed to vary with each iteration.

Using the gradient descent method, the update for the $(p-1)$th layer of a holomorphic complex-valued neural network is ([29], p. 60)

$$\Delta\mathbf{w}^{(p-1)} = -\left(\frac{\partial E}{\partial \mathbf{w}^{(p-1)}}\right)^*.$$

Suppose the activation function for the pth layer of the network, $p = 1, ..., L$, satisfies

$$\overline{g(z)} = g(\overline{z}).$$

Coordinate-wise the partial derivatives $\frac{\partial E}{\partial w_{lk}^{(L-1)}}$, taken with respect to the output

layer weights $w_{lk}^{(L-1)}$, $l = 1, ..., C$, $k = 1, ..., K_{L-1}$, are given by

$$\begin{aligned}
\frac{\partial E}{\partial w_{lk}^{(L-1)}} &= \frac{\partial}{\partial w_{lk}^{(L-1)}} \left[\frac{1}{N} \sum_{t=1}^{N} \sum_{h=1}^{C} (y_{th} - d_{th})(\overline{y_{th}} - \overline{y_{th}}) \right] \\
&= \frac{1}{N} \sum_{t=1}^{N} \left[\frac{\partial y_{tl}}{\partial w_{lk}^{(L-1)}} (\overline{y_{tl}} - \overline{d_{tl}}) + (y_{tl} - d_{tl}) \frac{\partial \overline{y_{tl}}}{\partial w_{lk}^{(L-1)}} \right] \\
&= \frac{1}{N} \sum_{t=1}^{N} \left(\frac{\partial y_{tl}}{\partial y_{tl}^{\text{net}}} \frac{\partial y_{tl}^{\text{net}}}{\partial w_{lk}^{(L-1)}} + \frac{\partial y_{tl}}{\partial \overline{y_{tl}^{\text{net}}}} \frac{\partial \overline{y_{tl}^{\text{net}}}}{\partial w_{lk}^{(L-1)}} \right) \left(\overline{y_{tl}} - \overline{d_{tl}} \right) \\
&= \frac{1}{N} \sum_{t=1}^{N} \left(\overline{y_{tl}} - \overline{d_{tl}} \right) g_L' \left(y_{tl}^{\text{net}} \right) x_{tk}^{(L-1)},
\end{aligned}$$

so that

$$\left(\frac{\partial E}{\partial w_{lk}^{(L-1)}} \right)^* = \frac{1}{N} \sum_{t=1}^{N} (y_{tl} - d_{tl}) g_L' \left(\overline{y_{tl}^{\text{net}}} \right) \overline{x_{tk}^{(L-1)}}.$$

The partial derivatives $\left(\frac{\partial E}{\partial w_{ji}^{(p-1)}} \right)^*$, taken with respect to the hidden layer weights $w_{ji}^{(p-1)}$, $1 \leq p \leq L-1$, $j = 1, ..., K_p$, $i = 1, ..., K_{p-1}$, are computed recursively. The partial derivatives $\frac{\partial E}{\partial w_{ji}^{(L-2)}}$, taken with respect to the $(L-2)$th hidden layer weights, are computed using the updated $(L-1)$th output layer weights:

$$\begin{aligned}
\frac{\partial E}{\partial w_{ji}^{(L-2)}} &= \frac{\partial}{\partial w_{ji}^{(L-2)}} \left[\frac{1}{N} \sum_{t=1}^{N} \sum_{l=1}^{C} (y_{tl} - d_{tl})(\overline{y_{tl}} - \overline{y_{tl}}) \right] \\
&= \frac{1}{N} \sum_{t=1}^{N} \sum_{l=1}^{C} \left[\frac{\partial y_{tl}}{\partial w_{ji}^{(L-2)}} (\overline{y_{tl}} - \overline{d_{tl}}) + (y_{tl} - d_{tl}) \frac{\partial \overline{y_{tl}}}{\partial w_{ji}^{(L-2)}} \right],
\end{aligned}$$

where

$$\begin{aligned}
\frac{\partial y_{tl}}{\partial w_{ji}^{(L-2)}} &= \frac{\partial y_{tl}}{\partial y_{tl}^{\text{net}}}\frac{\partial y_{tl}^{\text{net}}}{\partial w_{ji}^{(L-2)}} + \frac{\partial y_{tl}}{\partial \overline{y_{tl}^{\text{net}}}}\frac{\overline{\partial y_{tl}^{\text{net}}}}{\partial w_{ji}^{(L-2)}} \\
&= g_L'\left(y_{tl}^{\text{net}}\right)\left(\frac{\partial y_{tl}^{\text{net}}}{\partial x_{tj}^{(L-1)}}\frac{\partial x_{tj}^{(L-1)}}{\partial w_{ji}^{(L-2)}} + \frac{\partial y_{tl}^{\text{net}}}{\partial \overline{x_{tj}^{(L-1)}}}\frac{\partial \overline{x_{tj}^{(L-1)}}}{\partial w_{ji}^{(L-2)}}\right) \\
&= g_L'\left(y_{tl}^{\text{net}}\right) w_{lj}^{(L-1)}\left(\frac{\partial x_{tj}^{(L-1)}}{\partial\left(x_{tj}^{(L-1)}\right)^{\text{net}}}\frac{\partial\left(x_{tj}^{(L-1)}\right)^{\text{net}}}{\partial w_{ji}^{(L-2)}} + \frac{\partial x_{tj}^{(L-1)}}{\partial\overline{\left(x_{tj}^{(L-1)}\right)^{\text{net}}}}\frac{\partial\overline{\left(x_{tj}^{(L-1)}\right)^{\text{net}}}}{\partial w_{ji}^{(L-2)}}\right) \\
&= g_L'\left(y_{tl}^{\text{net}}\right) w_{lj}^{(L-1)} g_{L-1}'\left(\left(x_{tj}^{(L-1)}\right)^{\text{net}}\right) x_{ti}^{(L-2)}
\end{aligned}$$

and

$$\begin{aligned}
\frac{\partial \overline{y_{tl}}}{\partial w_{ji}^{(L-2)}} &= \frac{\partial \overline{y_{tl}}}{\partial \overline{y_{tl}^{\text{net}}}}\frac{\overline{\partial y_{tl}^{\text{net}}}}{\partial w_{ji}^{(L-2)}} + \frac{\partial \overline{y_{tl}}}{\partial y_{tl}^{\text{net}}}\frac{\partial y_{tl}^{\text{net}}}{\partial w_{ji}^{(L-2)}} \\
&= g_L'\left(\overline{y_{tl}^{\text{net}}}\right)\left(\frac{\partial \overline{y_{tl}^{\text{net}}}}{\partial \overline{x_{tj}^{(L-1)}}}\frac{\partial \overline{x_{tj}^{(L-1)}}}{\partial w_{ji}^{(L-2)}} + \frac{\partial \overline{y_{tl}^{\text{net}}}}{\partial x_{tj}^{(L-1)}}\frac{\partial x_{tj}^{(L-1)}}{\partial w_{ji}^{(L-2)}}\right) \\
&= g_L'\left(\overline{y_{tl}^{\text{net}}}\right)\overline{w_{lj}^{(L-1)}}\left(\frac{\partial \overline{x_{tj}^{(L-1)}}}{\partial\overline{\left(x_{tj}^{(L-1)}\right)^{\text{net}}}}\frac{\partial\overline{\left(x_{tj}^{(L-1)}\right)^{\text{net}}}}{\partial w_{ji}^{(L-2)}} + \frac{\partial \overline{x_{tj}^{(L-1)}}}{\partial\left(x_{tj}^{(L-1)}\right)^{\text{net}}}\frac{\partial\left(x_{tj}^{(L-1)}\right)^{\text{net}}}{\partial w_{ji}^{(L-2)}}\right) \\
&= 0,
\end{aligned}$$

so that

$$\left(\frac{\partial E}{\partial w_{ji}^{(L-2)}}\right)^* = \frac{1}{N}\sum_{t=1}^{N}\left(\sum_{l=1}^{C}(y_{tl}-d_{tl})g_L'\left(\overline{y_{tl}^{\text{net}}}\right)\overline{w_{lj}^{(L-1)}}\right)\cdot g_{L-1}'\left(\overline{\left(x_{tj}^{(L-1)}\right)^{\text{net}}}\right)\overline{x_{ti}^{(L-2)}},$$

and so on. We summarize the partial derivatives by

$$\left(\frac{\partial E}{\partial w_{ji}^{(p-1)}}\right)^* = \frac{1}{N}\sum_{t=1}^{N} E_{tj}^{(p)}\overline{x_{ti}^{(p-1)}}, \tag{2.3.3}$$

$1 \leq p \leq L$, where $j = 1, ..., K_p$, $i = 1, ..., K_{p-1}$, and the $E_{tj}^{(p)}$ are given recursively by

$$E_{tl}^{(L)} = (y_{tl} - d_{tl})\, g_L' \left(\overline{y_{tl}^{\text{net}}} \right), \tag{2.3.4}$$

where $l = 1, ..., C$, $t = 1, ..., N$; and for $1 \leq p \leq L - 1$,

$$E_{tj}^{(p)} = \left[\sum_{\alpha=1}^{K_{p+1}} E_{t\alpha}^{(p+1)} \overline{w_{\alpha j}^{(p)}} \right] g_p' \left(\overline{\left(x_{tj}^{(p)} \right)^{\text{net}}} \right), \tag{2.3.5}$$

where $j = 1, ..., K_p$, $t = 1, ..., N$. The gradient descent method is well known to be rather slow in the convergence of the error function. In the next chapter, we derive formulas for the backpropagation algorithm using Newton's method (compare with [9, 32]).

Chapter 3

The Newton's Method Backpropagation Algorithm

Newton's method has typically been avoided in neural network training due to computational inefficiency in computing and inverting the Hessian matrices. This difficulty is often due to the choice of activation function: for example, the so-called "split" activation functions provide for a very messy application of any minimization algorithm. The use of holomorphic activation functions allows for a ready application of Newton's method to neural network training algorithms. In this chapter, we develop a recursive algorithm given by Theorem 3.2.1 to efficiently compute the entries of the Hessian matrices in the implication of Newton's method to the backpropagation algorithm. Corollary 3.3.1 gives a similar recursive algorithm for computation of the Hessian matrices in the application of the pseudo-Newton method to the backpropagation algorithm based on Theorem 3.2.1. The recursive algorithms we develop are analogous to the known gradient descent backpropagation algorithm as stated in Chapter 2, hence can be readily implemented in real-world applications. A problem with Newton's method is the choice of steplengths to ensure the algorithm actually converges in applications. Our setting enables us to perform a rigorous analysis for a complex version of the one-step Newton steplength algorithm for the minimization of real-valued complex functions via Newton's method. This gives an adaptive learning rate algorithm to apply to such minimization, which is given in Theorem 3.4.4. We then apply our algorithms to train a small-scale

complex-valued MLP using the XOR data set, and compare the use of the Newton and pseudo-Newton methods to that of the standard gradient descent backpropagation algorithm. Our experiments show that the algorithms we develop use significantly fewer iterations to achieve the same results as the gradient descent algorithm. Newton's method thus provides a valuable tool for fast learning for complex-valued neural networks as a practical alternative to the gradient descent methods.

3.1 The $\mathbb{CR}$-Calculus and Newton's Method

We begin with a discussion of the $\mathbb{CR}$-calculus, which we use to compute the Hessian matrices for our application of Newton's method to the backpropagation algorithm. We employ the following notation from [29]. The cogradient (or $\mathbb{R}$-derivative) and conjugate cogradient (or $\overline{\mathbb{R}}$-derivative) operators on functions $\mathbb{C}^k \to \mathbb{C}^m$ are the row operators

$$\frac{\partial(\cdot)}{\partial \mathbf{z}} = \left(\frac{\partial(\cdot)}{\partial z_1}, ..., \frac{\partial(\cdot)}{\partial z_k} \right) \text{ and } \frac{\partial(\cdot)}{\partial \overline{\mathbf{z}}} = \left(\frac{\partial(\cdot)}{\partial \overline{z_1}}, ..., \frac{\partial(\cdot)}{\partial \overline{z_k}} \right),$$

respectively, where, writing $\mathbf{z} = \mathbf{x} + i\mathbf{y} \in \mathbb{C}^k$ with $\mathbf{x}, \mathbf{y} \in \mathbb{R}^k$ and denoting by $\frac{\partial}{\partial x_j}$ and $\frac{\partial}{\partial y_j}$ the partial derivative operators taken with respect to the real and imaginary parts of $z_j = x_j + iy_j$, respectively, the operators $\frac{\partial}{\partial z_j}$ and $\frac{\partial}{\partial \overline{z_j}}$ are defined by

$$\frac{\partial(\cdot)}{\partial z_j} := \frac{1}{2}\left(\frac{\partial(\cdot)}{\partial x_j} - i\frac{\partial(\cdot)}{\partial y_j} \right) \text{ and } \frac{\partial(\cdot)}{\partial \overline{z_j}} := \frac{1}{2}\left(\frac{\partial(\cdot)}{\partial x_j} + i\frac{\partial(\cdot)}{\partial y_j} \right)$$

for $j = 1, ..., k$. Let

$$\frac{\partial(\cdot)}{\partial \mathbf{x}} = \left(\frac{\partial(\cdot)}{\partial x_1}, ..., \frac{\partial(\cdot)}{\partial x_k} \right) \text{ and } \frac{\partial(\cdot)}{\partial \mathbf{y}} = \left(\frac{\partial(\cdot)}{\partial y_1}, ..., \frac{\partial(\cdot)}{\partial y_k} \right).$$

Set

$$J := \begin{bmatrix} I & iI \\ I & -iI \end{bmatrix} \in M_{2k\times 2k}(\mathbb{C}),$$

where I is the $k \times k$ identity matrix. For $\mathbf{z} = \mathbf{x} + i\mathbf{y} \in \mathbb{C}^k$ with $\mathbf{x}, \mathbf{y} \in \mathbb{R}^k$ we have the coordinate transformations

$$\begin{pmatrix} \mathbf{z} \\ \overline{\mathbf{z}} \end{pmatrix} = J \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \text{ and } \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \frac{1}{2} J^* \begin{pmatrix} \mathbf{z} \\ \overline{\mathbf{z}} \end{pmatrix} \tag{3.1.1}$$

and the cogradient transformations

$$\left(\frac{\partial(\cdot)}{\partial \mathbf{z}}, \frac{\partial(\cdot)}{\partial \overline{\mathbf{z}}} \right) = \frac{1}{2} \left(\frac{\partial(\cdot)}{\partial \mathbf{x}}, \frac{\partial(\cdot)}{\partial \mathbf{y}} \right) J^* \text{ and } \left(\frac{\partial(\cdot)}{\partial \mathbf{x}}, \frac{\partial(\cdot)}{\partial \mathbf{y}} \right) = \left(\frac{\partial(\cdot)}{\partial \mathbf{z}}, \frac{\partial(\cdot)}{\partial \overline{\mathbf{z}}} \right) J. \tag{3.1.2}$$

A function $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{C}^m$ is called real differentiable ($\mathbb{R}$-differentiable) if it is (Frechet) differentiable as a mapping

$$f(\mathbf{x}, \mathbf{y}) : D := \left\{ \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \in \mathbb{R}^{2k} \;\middle|\; \begin{array}{c} \mathbf{x}, \mathbf{y} \in \mathbb{R}^k, \\ \mathbf{z} = \mathbf{x} + i\mathbf{y} \in \Omega \end{array} \right\} \subseteq \mathbb{R}^{2k} \to \mathbb{R}^{2m}. \tag{3.1.3}$$

The partial derivatives $\frac{\partial f}{\partial \mathbf{z}}$ and $\frac{\partial f}{\partial \overline{\mathbf{z}}}$ exist if and only if f is $\mathbb{R}$-differentiable [29,44]. The distinction between $\mathbb{R}$-differentiability and complex differentiability is significant. In training a neural network, our goal is to minimize the real-valued error function as a function of the weights. Since we are using holomorphic activation functions, the error function is in fact a real-valued $\mathbb{R}$-differentiable function. Note that a nonconstant function $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ cannot be complex differentiable as it fails to satisfy the Cauchy-Riemann equations.

The updates for for the minimization of a function $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ using Newton's method are given by formula (111) of [29]:

$$\Delta \mathbf{z} = \left(\mathcal{H}_{\mathbf{zz}} - \mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}} \mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}^{-1} \mathcal{H}_{\mathbf{z}\overline{\mathbf{z}}} \right)^{-1} \left[\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}} \mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}^{-1} \left(\frac{\partial f}{\partial \overline{\mathbf{z}}} \right)^* - \left(\frac{\partial f}{\partial \mathbf{z}} \right)^* \right], \tag{3.1.4}$$

where

$$\mathcal{H}_{\mathbf{zz}} = \frac{\partial}{\partial \mathbf{z}} \left(\frac{\partial E}{\partial \mathbf{z}} \right)^* \text{ and } \mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}} = \frac{\partial}{\partial \overline{\mathbf{z}}} \left(\frac{\partial E}{\partial \mathbf{z}} \right)^*, \tag{3.1.5}$$

where the entries of $\left(\frac{\partial f}{\partial \mathbf{z}}\right)^*$ are given by (2.3.3). Note that although (3.2.1) asks for the four Hessian matrices $\mathcal{H}_{\mathbf{zz}}$, $\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}$, $\mathcal{H}_{\mathbf{z}\overline{\mathbf{z}}}$, and $\mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}$, we have $\mathcal{H}_{\mathbf{z}\overline{\mathbf{z}}} = \overline{\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}}$ and $\mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}} = \overline{\mathcal{H}_{\mathbf{zz}}}$.

Thus, to compute the Newton weight update, we need only compute the matrices given in 3.1.5.

As a special case of the complex Newton's method, the complex pseudo-Newton method takes $\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}} = \mathcal{H}_{\mathbf{z}\overline{\mathbf{z}}}$, reducing the weight updates to something similar to the real version of Newton's method:

$$\Delta \mathbf{z} = -\mathcal{H}_{\mathbf{zz}}^{-1} \left(\frac{\partial f}{\partial \mathbf{z}} \right)^* .$$

3.2 Backpropagation Using Newton's Method

In order to apply Newton's method to the minimization of the error function 2.2.1, we need to compute the weight updates for each layer of the network (we omit the superscripts, which index the layers, to simplify our writing):

$$\Delta \mathbf{w} = \left(\mathcal{H}_{\mathbf{ww}} - \mathcal{H}_{\overline{\mathbf{w}}\mathbf{w}} \mathcal{H}_{\overline{\mathbf{ww}}}^{-1} \mathcal{H}_{\mathbf{w}\overline{\mathbf{w}}} \right)^{-1} \left[\mathcal{H}_{\overline{\mathbf{w}}\mathbf{w}} \mathcal{H}_{\overline{\mathbf{ww}}}^{-1} \left(\frac{\partial E}{\partial \overline{\mathbf{w}}} \right)^* - \left(\frac{\partial E}{\partial \mathbf{w}} \right)^* \right]. \tag{3.2.1}$$

We consider the entries of the Hessian matrices $\mathcal{H}_{\mathbf{ww}}$ and $\mathcal{H}_{\overline{\mathbf{w}}\mathbf{w}}$. For the $(p-1)$th layer, the entries of $\mathcal{H}_{\mathbf{ww}}$ are given by (see (2.3.1))

$$\mathcal{H}_{\mathbf{ww}} \left[(j-1) \cdot K_{p-1} + i, (b-1) \cdot K_{p-1} + a \right] = \frac{\partial}{\partial w_{ba}^{(p-1)}} \left(\frac{\partial E}{\partial w_{ji}^{(p-1)}} \right)^* ,$$

where $j, b = 1, ..., K_p$ and $i, a = 1, ..., K_{p-1}$, and the entries of $\mathcal{H}_{\overline{\mathbf{w}}\mathbf{w}}$ are given by

$$\mathcal{H}_{\overline{\mathbf{w}}\mathbf{w}} \left[(j-1) \cdot K_{p-1} + i, (b-1) \cdot K_{p-1} + a \right] = \frac{\partial}{\partial \overline{w_{ba}^{(p-1)}}} \left(\frac{\partial E}{\partial w_{ji}^{(p-1)}} \right)^* ,$$

where $j, b = 1, ..., K_p$ and $i, a = 1, ..., K_{p-1}$.

First we derive an explicit formula for the entries of the Hessians $\mathcal{H}_{\mathbf{ww}}$. We start with the output layer and compute $\frac{\partial}{\partial w_{kq}^{(L-1)}} \left(\frac{\partial E}{\partial w_{lp}^{(L-1)}} \right)^*$, where $k, l = 1, ..., C$ and $q, p = 1, ..., K_{L-1}$. Observe that if $k \neq l$, then each term $(y_{tl} - d_{tl}) g_L' \left(\overline{y_{tl}^{\text{net}}} \right) \overline{x_{tp}^{(L-1)}}$ in the cogradient given by (2.3.3) and (2.3.4) does not depend on the weights $w_{kq}^{(L-1)}$, hence this entry of the Hessian will be 0. So the Hessian matrix for the output layer

has a block diagonal form:

$$\mathcal{H}_{\mathbf{w}^{(L-1)}\mathbf{w}^{(L-1)}} = \operatorname{diag}\left\{ \left[\frac{\partial}{\partial w_{lq}^{(L-1)}} \left(\frac{\partial E}{\partial w_{lp}^{(L-1)}} \right)^* \right]_{\substack{1\le p\le K_{L-1} \\ 1\le q\le K_{L-1}}} : l = 1, ..., C \right\}.$$

Now:

$$\begin{aligned} \frac{\partial}{\partial w_{lq}^{(L-1)}} \left(\frac{\partial E}{\partial w_{lp}^{(L-1)}} \right)^* &= \frac{\partial}{\partial w_{lq}^{(L-1)}} \left[\frac{1}{N} \sum_{t=1}^{N} (y_{tl} - d_{tl}) g_L' \left(\overline{y_{tl}^{\text{net}}} \right) \overline{x_{tp}^{(L-1)}} \right] \\ &= \frac{1}{N} \sum_{t=1}^{N} \left[(y_{tl} - d_{tl}) \frac{\partial g_L' \left(\overline{y_{tl}^{\text{net}}} \right)}{\partial w_{lq}^{(L-1)}} + g_L' \left(\overline{y_{tl}^{\text{net}}} \right) \frac{\partial y_{tl}}{\partial w_{lq}^{(L-1)}} \right] \overline{x_{tp}^{(L-1)}} \end{aligned} \tag{3.2.2}$$

where

$$\frac{\partial y_{tl}}{\partial w_{lq}^{(L-1)}} = \frac{\partial y_{tl}}{\partial y_{tl}^{\text{net}}} \frac{\partial y_{tl}^{\text{net}}}{\partial w_{lq}^{(L-1)}} + \frac{\partial y_{tl}}{\partial \overline{y_{tl}^{\text{net}}}} \frac{\partial \overline{y_{tl}^{\text{net}}}}{\partial w_{lq}^{(L-1)}} = g_L' \left(y_{tl}^{\text{net}} \right) x_{tq}^{(L-1)}$$

since g_L is holomorphic and therefore $\frac{\partial y_{tl}}{\partial \overline{y_{tl}^{\text{net}}}} = 0$ (Cauchy-Riemann condition), and similarly

$$\frac{\partial g_L' \left(\overline{y_{tl}^{\text{net}}} \right)}{\partial w_{lq}^{(L-1)}} = \frac{\partial g_L' \left(\overline{y_{tl}^{\text{net}}} \right)}{\partial \overline{y_{tl}^{\text{net}}}} \frac{\overline{\partial y_{tl}^{\text{net}}}}{\partial w_{lq}^{(L-1)}} + \frac{\partial g_L' \left(\overline{y_{tl}^{\text{net}}} \right)}{\partial y_{tl}^{\text{net}}} \frac{\partial y_{tl}^{\text{net}}}{\partial w_{lq}^{(L-1)}} = 0.$$

Combining these two partial derivatives with (3.2.2) gives the following formula for the entries of the output layer Hessian matrix:

$$\frac{\partial}{\partial w_{kq}^{(L-1)}} \left(\frac{\partial E}{\partial w_{lp}^{(L-1)}} \right)^* = \begin{cases} \frac{1}{N} \sum_{t=1}^{N} g_L' \left(\overline{y_{tl}^{\text{net}}} \right) g_L' \left(y_{tl}^{\text{net}} \right) \overline{x_{tp}^{(L-1)}} x_{tq}^{(L-1)} & \text{if } k = l, \\ 0 & \text{if } k \neq l. \end{cases} \tag{3.2.3}$$

After updating the output layer weights, the backpropagation algorithm updates the hidden layer weights recursively. We compute the entries of the Hessian

$\mathcal{H}_{\mathbf{w}^{(p-1)}\mathbf{w}^{(p-1)}}$ for the $(p-1)$th layer using (2.3.3):

$$\frac{\partial}{\partial w_{ba}^{(p-1)}}\left(\frac{\partial E}{\partial w_{ji}^{(p-1)}}\right)^{*}=\frac{\partial}{\partial w_{ba}^{(p-1)}}\left[\frac{1}{N}\sum_{t=1}^{N}E_{tj}^{(p)}\overline{x_{ti}^{(p-1)}}\right]=\frac{1}{N}\sum_{t=1}^{N}\frac{\partial E_{tj}^{(p)}}{\partial w_{ba}^{(p-1)}}\overline{x_{ti}^{(p-1)}}. \tag{3.2.4}$$

Applying the chain rule to (2.3.5), we have

$$\begin{aligned}\frac{\partial E_{tj}^{(p)}}{\partial w_{ba}^{(p-1)}}&=\frac{\partial}{\partial w_{ba}^{(p-1)}}\left[\left(\sum_{\eta=1}^{K_{p+1}}E_{t\eta}^{(p+1)}\overline{w_{\eta j}^{(p)}}\right)g_p'\left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}}\right)\right]\\&=g_p'\left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}}\right)\sum_{\eta=1}^{K_{p+1}}\frac{\partial E_{t\eta}^{(p+1)}}{\partial w_{ba}^{(p-1)}}\overline{w_{\eta j}^{(p)}}\\&=g_p'\left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}}\right)\sum_{\eta=1}^{K_{p+1}}\left[\frac{\partial E_{t\eta}^{(p+1)}}{\partial x_{tb}^{(p)}}\frac{\partial x_{tb}^{(p)}}{\partial w_{ba}^{(p-1)}}+\frac{\partial E_{t\eta}^{(p+1)}}{\partial\overline{x_{tb}^{(p)}}}\frac{\partial\overline{x_{tb}^{(p)}}}{\partial w_{ba}^{(p-1)}}\right]\overline{w_{\eta j}^{(p)}}\\&=g_p'\left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}}\right)\sum_{\eta=1}^{K_{p+1}}\frac{\partial E_{t\eta}^{(p+1)}}{\partial x_{tb}^{(p)}}\left[\frac{\partial x_{tb}^{(p)}}{\partial(x_{tb}^{(p)})^{\text{net}}}\frac{\partial(x_{tb}^{(p)})^{\text{net}}}{\partial w_{ba}^{(p-1)}}+\frac{\partial x_{tb}^{(p)}}{\partial\overline{(x_{tb}^{(p)})^{\text{net}}}}\frac{\partial\overline{(x_{tb}^{(p)})^{\text{net}}}}{\partial w_{ba}^{(p-1)}}\right]\overline{w_{\eta j}^{(p)}}\\&=g_p'\left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}}\right)\sum_{\eta=1}^{K_{p+1}}\frac{\partial E_{t\eta}^{(p+1)}}{\partial x_{tb}^{(p)}}g_p'\left(\left(x_{tb}^{(p)}\right)^{\text{net}}\right)x_{ta}^{(p-1)}\overline{w_{\eta j}^{(p)}}.\end{aligned} \tag{3.2.5}$$

In the above computation, we have used the fact that g_p is holomorphic and hence $\frac{\partial g_p'\left(\overline{(x_{tj}^{(p)})^{\text{net}}}\right)}{\partial w_{ba}^{(p-1)}}=0$ and $\frac{\partial\overline{x_{tb}^{(p)}}}{\partial w_{ba}^{(p-1)}}=0$, $\frac{\partial x_{tb}^{(p)}}{\partial\overline{(x_{tb}^{(p)})^{\text{net}}}}=0$, and $\frac{\partial\overline{(x_{tb}^{(p)})^{\text{net}}}}{\partial w_{ba}^{(p-1)}}=0$. Combining (3.2.4) and (3.2.5), we have:

$$\begin{aligned}\frac{\partial}{\partial w_{ba}^{(p-1)}}\left(\frac{\partial E}{\partial w_{ji}^{(p-1)}}\right)^{*}=\frac{1}{N}\sum_{t=1}^{N}&\left[\sum_{\eta=1}^{K_{p+1}}\frac{\partial E_{t\eta}^{(p+1)}}{\partial x_{tb}^{(p)}}\overline{w_{\eta j}^{(p)}}\right]\\&\cdot g_p'\left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}}\right)g_p'\left(\left(x_{tb}^{(p)}\right)^{\text{net}}\right)\overline{x_{ti}^{(p-1)}}x_{ta}^{(p-1)}.\end{aligned} \tag{3.2.6}$$

Next, we derive a recursive rule for finding the partial derivatives $\frac{\partial E_{t\eta}^{(p+1)}}{\partial x_{tb}^{(p)}}$. For

computational purposes, an explicit formula for $\frac{\partial E_{t\eta}^{(L)}}{\partial x_{tb}^{(L-1)}}$ is not necessary. What we need is a recursive formula for these partial derivatives as will be apparent shortly. Using (2.3.5) we have the following:

$$\begin{aligned}
\frac{\partial E_{t\eta}^{(p+1)}}{\partial x_{tb}^{(p)}} &= \frac{\partial}{\partial x_{tb}^{(p)}}\left[\left(\sum_{\alpha=1}^{K_{p+2}} E_{t\alpha}^{(p+2)}\overline{w_{\alpha\eta}^{(p+1)}}\right) g'_{p+1}\left(\overline{\left(x_{t\eta}^{(p+1)}\right)^{\text{net}}}\right)\right] \\
&= g'_{p+1}\left(\overline{\left(x_{t\eta}^{(p+1)}\right)^{\text{net}}}\right)\sum_{\alpha=1}^{K_{p+2}}\frac{\partial E_{t\alpha}^{(p+2)}}{\partial x_{tb}^{(p)}}\overline{w_{\alpha\eta}^{(p+1)}} \\
&= g'_{p+1}\left(\overline{\left(x_{t\eta}^{(p+1)}\right)^{\text{net}}}\right)\sum_{\alpha=1}^{K_{p+2}}\sum_{\beta=1}^{K_{p+1}}\left[\frac{\partial E_{t\alpha}^{(p+2)}}{\partial x_{t\beta}^{(p+1)}}\frac{\partial x_{t\beta}^{(p+1)}}{\partial x_{tb}^{(p)}} + \frac{\partial E_{t\alpha}^{(p+2)}}{\partial \overline{x_{t\beta}^{(p+1)}}}\frac{\partial \overline{x_{t\beta}^{(p+1)}}}{\partial x_{tb}^{(p)}}\right]\overline{w_{\alpha\eta}^{(p+1)}} \\
&= g'_{p+1}\left(\overline{\left(x_{t\eta}^{(p+1)}\right)^{\text{net}}}\right)\sum_{\alpha=1}^{K_{p+2}}\sum_{\beta=1}^{K_{p+1}}\frac{\partial E_{t\alpha}^{(p+2)}}{\partial x_{t\beta}^{(p+1)}} \\
&\qquad\cdot\left[\frac{\partial x_{t\beta}^{(p+1)}}{\partial (x_{t\beta}^{(p+1)})^{\text{net}}}\frac{\partial (x_{t\beta}^{(p+1)})^{\text{net}}}{\partial x_{tb}^{(p)}} + \frac{\partial x_{t\beta}^{(p+1)}}{\partial \overline{(x_{t\beta}^{(p+1)})^{\text{net}}}}\frac{\partial \overline{(x_{t\beta}^{(p+1)})^{\text{net}}}}{\partial x_{tb}^{(p)}}\right]\overline{w_{\alpha\eta}^{(p+1)}} \\
&= \sum_{\beta=1}^{K_{p+1}}\left[\sum_{\alpha=1}^{K_{p+2}}\frac{\partial E_{t\alpha}^{(p+2)}}{\partial x_{t\beta}^{(p+1)}}\overline{w_{\alpha\eta}^{(p+1)}}\right] g'_{p+1}\left(\overline{\left(x_{t\eta}^{(p+1)}\right)^{\text{net}}}\right) g'_{p+1}\left(\left(x_{t\beta}^{(p+1)}\right)^{\text{net}}\right) w_{\beta b}^{(p)}.
\end{aligned} \tag{3.2.7}$$

This gives a recursive formula for computing the partial derivatives $\frac{\partial E_{t\eta}^{(p+1)}}{\partial x_{tb}^{(p)}}$. We will combine the above calculations to give a more concise recursive algorithm for computing the entries of the matrices $\mathcal{H}_{\mathbf{ww}}$ in Theorem 3.2.1, below.

Next we consider the Hessians $\mathcal{H}_{\overline{\mathbf{w}}\mathbf{w}}$. Again we start with the output layer and compute $\frac{\partial}{\partial \overline{w_{kq}^{(L-1)}}}\left(\frac{\partial E}{\partial w_{lp}^{(L-1)}}\right)^*$. Using the fact that $\frac{\partial E}{\partial w_{lp}^{(L-1)}}$ does not depend on $\overline{w_{kq}^{(L-1)}}$ if $k \neq l$, we see that the output layer Hessian $\mathcal{H}_{\overline{\mathbf{w}^{(L-1)}}\mathbf{w}^{(L-1)}}$ is also block diagonal with blocks

$$\left[\frac{\partial}{\partial \overline{w_{lq}^{(L-1)}}}\left(\frac{\partial E}{\partial w_{lp}^{(L-1)}}\right)^*\right]_{\substack{1\le p\le K_{L-1}\\ 1\le q\le K_{L-1}}}$$

for $l = 1, ..., C$. Computing the entries in these blocks,

$$\frac{\partial}{\partial \overline{w_{lq}^{(L-1)}}} \left(\frac{\partial E}{\partial w_{lp}^{(L-1)}} \right)^* = \frac{\partial}{\partial \overline{w_{lq}^{(L-1)}}} \left[\frac{1}{N} \sum_{t=1}^{N} (y_{tl} - d_{tl}) g_L' \left(\overline{y_{tl}^{\text{net}}} \right) \overline{x_{tp}^{(L-1)}} \right]$$

$$= \frac{1}{N} \sum_{t=1}^{N} \left[(y_{tl} - d_{tl}) \frac{\partial g_L' \left(\overline{y_{tl}^{\text{net}}} \right)}{\partial \overline{w_{lq}^{(L-1)}}} + g_L' \left(\overline{y_{tl}^{\text{net}}} \right) \frac{\partial y_{tl}}{\partial \overline{w_{lq}^{(L-1)}}} \right] \overline{x_{tp}^{(L-1)}}$$

where $\frac{\partial y_{tl}}{\partial \overline{w_{lq}^{(L-1)}}} = 0$, and

$$\frac{\partial g_L' \left(\overline{y_{tl}^{\text{net}}} \right)}{\partial \overline{w_{lq}^{(L-1)}}} = \frac{\partial g_L' \left(\overline{y_{tl}^{\text{net}}} \right)}{\partial \overline{y_{tl}^{\text{net}}}} \frac{\partial \overline{y_{tl}^{\text{net}}}}{\partial \overline{w_{lq}^{(L-1)}}} + \frac{\partial g_L' \left(\overline{y_{tl}^{\text{net}}} \right)}{\partial y_{tl}^{\text{net}}} \frac{\partial y_{tl}^{\text{net}}}{\partial \overline{w_{lq}^{(L-1)}}} = g_L'' \left(\overline{y_{tl}^{\text{net}}} \right) \overline{x_{tq}^{(L-1)}}.$$

Thus:

$$\frac{\partial}{\partial \overline{w_{kq}^{(L-1)}}} \left(\frac{\partial E}{\partial w_{lp}^{(L-1)}} \right)^* = \begin{cases} \frac{1}{N} \sum_{t=1}^{N} (y_{tl} - d_{tl}) g_L'' \left(\overline{y_{tl}^{\text{net}}} \right) \overline{x_{tq}^{(L-1)} x_{tp}^{(L-1)}} & \text{if } k = l, \\ 0 & \text{if } k \neq l. \end{cases} \tag{3.2.8}$$

The entries of the Hessian $\mathcal{H}_{\overline{\mathbf{w}^{(p-1)}} \mathbf{w}^{(p-1)}}$ for the $(p-1)$th layer can be computed similarly. Using the cogradients (2.3.3) we have:

$$\frac{\partial}{\partial \overline{w_{ba}^{(p-1)}}} \left(\frac{\partial E}{\partial w_{ji}^{(p-1)}} \right)^* = \frac{1}{N} \sum_{t=1}^{N} \frac{\partial E_{tj}^{(p)}}{\partial \overline{w_{ba}^{(p-1)}}} \overline{x_{ti}^{(p-1)}}, \tag{3.2.9}$$

where $j, b = 1, ..., K_p$ and $i, a = 1, ..., K_{p-1}$. Using (2.3.5),

$$\frac{\partial E_{tj}^{(p)}}{\partial \overline{w_{ba}^{(p-1)}}} = \frac{\partial}{\partial \overline{w_{ba}^{(p-1)}}} \left[\left(\sum_{\eta=1}^{K_{p+1}} E_{t\eta}^{(p+1)} \overline{w_{\eta j}^{(p)}} \right) g_p' \left(\overline{\left(x_{tj}^{(p)} \right)^{\text{net}}} \right) \right]$$

$$= g_p' \left(\overline{\left(x_{tj}^{(p)} \right)^{\text{net}}} \right) \sum_{\eta=1}^{K_{p+1}} \frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{w_{ba}^{(p-1)}}} \overline{w_{\eta j}^{(p)}} + \frac{\partial g_p' \left(\overline{\left(x_{tj}^{(p)} \right)^{\text{net}}} \right)}{\partial \overline{w_{ba}^{(p-1)}}} \sum_{\eta=1}^{K_{p+1}} E_{t\eta}^{(p+1)} \overline{w_{\eta j}^{(p)}},$$

where

$$\begin{aligned}\frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{w_{ba}^{(p-1)}}} &= \frac{\partial E_{t\eta}^{(p+1)}}{\partial x_{tb}^{(p)}} \frac{\partial x_{tb}^{(p)}}{\partial \overline{w_{ba}^{(p-1)}}} + \frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}} \frac{\overline{\partial x_{tb}^{(p)}}}{\partial \overline{w_{ba}^{(p-1)}}} \\ &= \frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}} \left[\frac{\partial \overline{x_{tb}^{(p)}}}{\partial \overline{\left(x_{tb}^{(p)}\right)^{\text{net}}}} \frac{\partial \overline{\left(x_{tb}^{(p)}\right)^{\text{net}}}}{\partial \overline{w_{ba}^{(p-1)}}} + \frac{\partial \overline{x_{tb}^{(p)}}}{\partial \left(x_{tb}^{(p)}\right)^{\text{net}}} \frac{\partial \left(x_{tb}^{(p)}\right)^{\text{net}}}{\partial \overline{w_{ba}^{(p-1)}}} \right] \\ &= \frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}} g_p' \left(\overline{\left(x_{tb}^{(p)}\right)^{\text{net}}} \right) \overline{x_{ta}^{(p-1)}}\end{aligned}$$

and

$$\begin{aligned}\frac{\partial g_p' \left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}} \right)}{\partial \overline{w_{ba}^{(p-1)}}} &= \frac{\partial g_p' \left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}} \right)}{\partial \overline{\left(x_{tj}^{(p)}\right)^{\text{net}}}} \frac{\partial \overline{\left(x_{tj}^{(p)}\right)^{\text{net}}}}{\partial \overline{w_{ba}^{(p-1)}}} + \frac{\partial g_p' \left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}} \right)}{\partial \left(x_{tj}^{(p)}\right)^{\text{net}}} \frac{\left(x_{tj}^{(p)}\right)^{\text{net}}}{\partial \overline{w_{ba}^{(p-1)}}} \\ &= \begin{cases} g_p'' \left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}} \right) \overline{x_{ta}^{(p-1)}} & \text{if } j = b, \\ 0 & \text{if } j \neq b, \end{cases}\end{aligned}$$

so that

$$\frac{\partial E_{tj}^{(p)}}{\partial \overline{w_{ba}^{(p-1)}}} = \begin{cases} \left\{ \left[\sum_{\eta=1}^{K_{p+1}} \frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}} \overline{w_{\eta j}^{(p)}} \right] g_p' \left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}} \right) g_p' \left(\overline{\left(x_{tb}^{(p)}\right)^{\text{net}}} \right) \right. \\ \qquad \left. + \left[\sum_{\eta=1}^{K_{p+1}} E_{t\eta}^{(p+1)} \overline{w_{\eta j}^{(p)}} \right] g_p'' \left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}} \right) \right\} \overline{x_{ta}^{(p-1)}} & \text{if } j = b, \\ \left[\sum_{\eta=1}^{K_{p+1}} \frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}} \overline{w_{\eta j}^{(p)}} \right] g_p' \left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}} \right) g_p' \left(\overline{\left(x_{tb}^{(p)}\right)^{\text{net}}} \right) \overline{x_{ta}^{(p-1)}} & \text{if } j \neq b. \end{cases} \tag{3.2.10}$$

Combining (3.2.9) and (3.2.10), we get

$$\frac{\partial}{\partial \overline{w_{ba}^{(p-1)}}}\left(\frac{\partial E}{\partial w_{ji}^{(p-1)}}\right)^{*} = \begin{cases} \frac{1}{N}\sum_{t=1}^{N}\left\{\left[\sum_{\eta=1}^{K_{p+1}}\frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}}\overline{w_{\eta j}^{(p)}}\right] g_p'(\overline{(x_{tj}^{(p)})^{\text{net}}}) g_p'(\overline{(x_{tb}^{(p)})^{\text{net}}}) \right. \\ \qquad \left. + \left[\sum_{\eta=1}^{K_{p+1}} E_{t\eta}^{(p+1)} w_{\eta j}^{(p)}\right] g_p''(\overline{(x_{tj}^{(p)})^{\text{net}}})\right\} \overline{x_{ti}^{(p-1)} x_{ta}^{(p-1)}} & \text{if } j = b, \\ \frac{1}{N}\sum_{t=1}^{N}\left\{\left[\sum_{\eta=1}^{K_{p+1}}\frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}}\overline{w_{\eta j}^{(p)}}\right] g_p'(\overline{(x_{tj}^{(p)})^{\text{net}}}) g_p'(\overline{(x_{tb}^{(p)})^{\text{net}}})\right\} \\ \qquad \cdot \overline{x_{ti}^{(p-1)} x_{ta}^{(p-1)}} & \text{if } j \neq b, \end{cases} \tag{3.2.11}$$

where $j, b = 1, ..., K_p$ and $i, a = 1, ..., K_{p+1}$, and the partial derivatives $\frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}}$ can be computed recursively:

$$\frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}} = \frac{\partial}{\partial \overline{x_{tb}^{(p)}}} \left[\left(\sum_{\alpha=1}^{K_{p+2}} E_{t\alpha}^{(p+2)} \overline{w_{\alpha\eta}^{(p+1)}} \right) g'_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right) \right]$$

$$= g'_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right) \sum_{\alpha=1}^{K_{p+2}} \frac{\partial E_{t\alpha}^{(p+2)}}{\partial \overline{x_{tb}^{(p)}}} \overline{w_{\alpha\eta}^{(p+1)}} + \frac{\partial g'_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right)}{\partial \overline{x_{tb}^{(p)}}} \sum_{\alpha=1}^{K_{p+2}} E_{t\alpha}^{(p+2)} \overline{w_{\alpha\eta}^{(p+1)}}$$

$$= g'_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right) \sum_{\alpha=1}^{K_{p+2}} \sum_{\beta=1}^{K_{p+1}} \left[\frac{\partial E_{t\alpha}^{(p+2)}}{\partial x_{t\beta}^{(p+1)}} \frac{\partial x_{t\beta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}} + \frac{\partial E_{t\alpha}^{(p+2)}}{\partial \overline{x_{t\beta}^{(p+1)}}} \frac{\partial \overline{x_{t\beta}^{(p+1)}}}{\partial \overline{x_{tb}^{(p)}}} \right] \overline{w_{\alpha\eta}^{(p+1)}}$$

$$+ \left[\frac{\partial g'_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right)}{\partial \overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}}} \frac{\partial \overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}}}{\partial \overline{x_{tb}^{(p)}}} \right.$$

$$\left. + \frac{\partial g'_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right)}{\partial \left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \frac{\partial \left(x_{t\eta}^{(p+1)} \right)^{\text{net}}}{\partial \overline{x_{tb}^{(p)}}} \right] \sum_{\alpha=1}^{K_{p+2}} E_{t\alpha}^{(p+2)} \overline{w_{\alpha\eta}^{(p+1)}}$$

$$= g'_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right) \sum_{\alpha=1}^{K_{p+2}} \sum_{\beta=1}^{K_{p+1}} \frac{\partial E_{t\alpha}^{(p+2)}}{\partial \overline{x_{t\beta}^{(p+1)}}} \left[\frac{\partial \overline{x_{t\beta}^{(p+1)}}}{\partial \left(x_{t\beta}^{(p+1)} \right)^{\text{net}}} \frac{\partial \left(x_{t\beta}^{(p+1)} \right)^{\text{net}}}{\partial \overline{x_{tb}^{(p)}}} \right.$$

$$\left. + \frac{\partial \overline{x_{t\beta}^{(p+1)}}}{\partial \overline{\left(x_{t\beta}^{(p+1)} \right)^{\text{net}}}} \frac{\partial \overline{\left(x_{t\beta}^{(p+1)} \right)^{\text{net}}}}{\partial \overline{x_{tb}^{(p)}}} \right] \overline{w_{\alpha\eta}^{(p+1)}}$$

$$+ g''_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right) \overline{w_{\eta b}^{(p)}} \sum_{\alpha=1}^{K_{p+2}} E_{t\alpha}^{(p+2)} \overline{w_{\alpha\eta}^{(p+1)}}$$

$$= g'_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right) \sum_{\alpha=1}^{K_{p+2}} \sum_{\beta=1}^{K_{p+1}} \frac{\partial E_{t\alpha}^{(p+2)}}{\partial \overline{x_{t\beta}^{(p+1)}}} g'_{p+1} \left(\overline{\left(x_{t\beta}^{(p+1)} \right)^{\text{net}}} \right) \overline{w_{\beta b}^{(p)} w_{\alpha\eta}^{(p+1)}}$$

$$+ g''_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right) \overline{w_{\eta b}^{(p)}} \sum_{\alpha=1}^{K_{p+2}} E_{t\alpha}^{(p+2)} \overline{w_{\alpha\eta}^{(p+1)}}$$

$$= \sum_{\beta=1}^{K_{p+1}} \left[\sum_{\alpha=1}^{K_{p+2}} \frac{\partial E_{t\alpha}^{(p+2)}}{\partial \overline{x_{t\beta}^{(p+1)}}} \overline{w_{\alpha\eta}^{(p+1)}} \right] g'_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right) g'_{p+1} \left(\overline{\left(x_{t\beta}^{(p+1)} \right)^{\text{net}}} \right) \overline{w_{\beta b}^{(p)}}$$

$$+ \left[\sum_{\alpha=1}^{K_{p+2}} E_{t\alpha}^{(p+2)} \overline{w_{\alpha\eta}^{(p+1)}} \right] g''_{p+1} \left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right) \overline{w_{\eta b}^{(p)}}.$$

To summarize the above calculation, we have

$$\frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}} = \sum_{\beta=1}^{K_{p+1}} \left[\sum_{\alpha=1}^{K_{p+2}} \frac{\partial E_{t\alpha}^{(p+2)}}{\partial \overline{x_{t\beta}^{(p+1)}}} \overline{w_{\alpha\eta}^{(p+1)}} \right] g'_{p+1}\left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right) g'_{p+1}\left(\overline{\left(x_{t\beta}^{(p+1)} \right)^{\text{net}}} \right) \overline{w_{\beta b}^{(p)}}$$
$$+ \left[\sum_{\alpha=1}^{K_{p+2}} E_{t\alpha}^{(p+2)} \overline{w_{\alpha\eta}^{(p+1)}} \right] g''_{p+1}\left(\overline{\left(x_{t\eta}^{(p+1)} \right)^{\text{net}}} \right) \overline{w_{\eta b}^{(p)}}. \tag{3.2.12}$$

We now summarize the formulas we have derived in the following theorem.

Theorem 3.2.1 (Newton Backpropagation Algorithm for Holomorphic Neural Networks, [30], Theorem 4.1)**.** *The weight updates for the holomorphic MLPs with activation functions satisfying*

$$\overline{g(z)} = g(\overline{z}),$$

$p = 1, ..., L$, using the backpropagation algorithm with Newton's method are given by

$$\Delta \mathbf{w}^{(p-1)} = \left(\mathcal{H}_{\mathbf{w}^{(p-1)}\mathbf{w}^{(p-1)}} - \mathcal{H}_{\overline{\mathbf{w}^{(p-1)}}\mathbf{w}^{(p-1)}} \mathcal{H}^{-1}_{\overline{\mathbf{w}^{(p-1)}}\mathbf{w}^{(p-1)}} \mathcal{H}_{\mathbf{w}^{(p-1)}\overline{\mathbf{w}^{(p-1)}}} \right)^{-1}$$
$$\cdot \left[\mathcal{H}_{\overline{\mathbf{w}^{(p-1)}}\mathbf{w}^{(p-1)}} \mathcal{H}^{-1}_{\overline{\mathbf{w}^{(p-1)}}\mathbf{w}^{(p-1)}} \left(\frac{\partial E}{\partial \overline{\mathbf{w}^{(p-1)}}} \right)^{*} - \left(\frac{\partial E}{\partial \mathbf{w}^{(p-1)}} \right)^{*} \right], \tag{3.2.13}$$

where:

1. *the entries of the Hessian matrices $\mathcal{H}_{\mathbf{w}^{(p-1)}\mathbf{w}^{(p-1)}}$ for $p = 1, ..., L$ are given by*

$$\frac{\partial}{\partial w_{ba}^{(p-1)}} \left(\frac{\partial E}{\partial w_{ji}^{(p-1)}} \right)^{*} = \frac{1}{N} \sum_{t=1}^{N} \gamma_{tjb}^{(p)} \overline{x_{ti}^{(p-1)}} x_{ta}^{(p-1)} \tag{3.2.14}$$

for $j, b = 1, ..., K_p$ and $i, a = 1, ..., K_{p-1}$, where the $\gamma_{tjb}^{(p)}$ are defined for $t = 1, ..., N$ recursively on p by

$$\gamma_{tkl}^{(L)} = \begin{cases} g'_L(\overline{y_{tl}^{net}}) g'_L(y_{tl}^{net}) & \text{if } k = l, \\ 0 & \text{if } k \neq l, \end{cases}$$

for $k, l = 1, ..., C$, *and for* $p = 1, ..., L-1$,

$$\gamma_{tjb}^{(p)} = \left[\sum_{\eta=1}^{K_{p+1}} \sum_{\beta=1}^{K_{p+1}} \gamma_{t\eta\beta}^{(p+1)} \overline{w_{\eta j}^{(p)}} w_{\beta b}^{(p)} \right] g_p' \left(\overline{(x_{tj}^{(p)})^{net}} \right) g_p' \left((x_{tb}^{(p)})^{net} \right) \tag{3.2.15}$$

for $j, b = 1, ..., K_{p+1}$,

2. *the entries of the Hessian matrices* $\mathcal{H}_{\overline{\mathbf{w}^{(p-1)}}\mathbf{w}^{(p-1)}}$ *for* $p = 1, ..., L$ *are given by*

$$\frac{\partial}{\partial \overline{w_{ba}^{(p-1)}}} \left(\frac{\partial E}{\partial w_{ji}^{(p-1)}} \right)^* = \frac{1}{N} \sum_{t=1}^{N} \left(\psi_{tjb}^{(p)} + \theta_{tjb}^{(p)} \right) \overline{x_{ti}^{(p-1)} x_{ta}^{(p-1)}} \tag{3.2.16}$$

for $j, b = 1, ..., K_p$ *and* $i, a = 1, ..., K_{p-1}$, *where the* $\theta_{tjb}^{(p)}$ *are defined for* $t = 1, ..., N$ *by*

$$\theta_{tkl}^{(L)} = \begin{cases} (y_{tl} - d_{tl}) g_L'' \left(\overline{y_{tl}^{net}} \right) & \text{if } k = l, \\ 0 & \text{if } k \neq l, \end{cases}$$

for $k, l = 1, ..., C$, *and for* $p = 1, ..., L-1$,

$$\theta_{tjb}^{(p)} = \begin{cases} \left[\sum_{\eta=1}^{K_{p+1}} E_{t\eta}^{(p+1)} w_{\eta j}^{(p)} \right] g_p'' \left(\overline{\left(x_{tj}^{(p)} \right)^{net}} \right) & \text{if } j = b, \\ 0 & \text{if } j \neq b, \end{cases} \tag{3.2.17}$$

for $j, b = 1, ..., K_{p+1}$, *where the* $E_{t\eta}^{(p)}$ *are given by (2.3.4) and (2.3.5), and the* $\psi_{tjb}^{(p)}$ *are defined for* $t = 1, ..., N$ *recursively on* p *by* $\psi_{tkl}^{(L)} = 0$ *for* $k, l = 1, ..., C$, *and for* $p = 1, ..., L-1$,

$$\psi_{tjb}^{(p)} = \left[\sum_{\eta=1}^{K_{p+1}} \sum_{\beta=1}^{K_{p+1}} \left(\psi_{t\eta\beta}^{(p+1)} \overline{w_{\beta b}^{(p)}} + \theta_{t\eta\beta}^{(p+1)} \overline{w_{\eta b}^{(p)}} \right) \overline{w_{\eta j}^{(p)}} \right] g_p' \left(\overline{\left(x_{tj}^{(p)} \right)^{net}} \right) g_p' \left(\overline{\left(x_{tb}^{(p)} \right)^{net}} \right) \tag{3.2.18}$$

for $j, b = 1, ..., K_{p+1}$, *and*

3. *for the other two Hessian matrices we have* $\mathcal{H}_{\mathbf{w}^{(p-1)}\overline{\mathbf{w}^{(p-1)}}} = \overline{\mathcal{H}_{\overline{\mathbf{w}^{(p-1)}}\mathbf{w}^{(p-1)}}}$ *and* $\mathcal{H}_{\overline{\mathbf{w}^{(p-1)}\mathbf{w}^{(p-1)}}} = \overline{\mathcal{H}_{\mathbf{w}^{(p-1)}\mathbf{w}^{(p-1)}}}$.

Proof. 1. Setting $\gamma_{tkl}^{(L)}$ as defined above, Equation (3.2.14) follows immediately from (3.2.3). For the hidden layer Hessian matrix entries, set

$$\gamma_{tjb}^{(p)} = \left[\sum_{\eta=1}^{K_{p+1}} \frac{\partial E_{t\eta}^{(p+1)}}{\partial x_{tb}^{(p)}} \overline{w_{\eta j}^{(p)}}\right] g_p'\left(\overline{(x_{tj}^{(p)})^{\text{net}}}\right) g_p'\left((x_{tb}^{(p)})^{\text{net}}\right) \tag{3.2.19}$$

in (3.2.6), giving us (3.2.14). Then using (3.2.7) we have

$$\frac{\partial E_{t\eta}^{(p+1)}}{\partial x_{tb}^{(p)}} = \sum_{\beta=1}^{K_{p+1}} \gamma_{t\eta\beta}^{(p+1)} w_{\beta b}^{(p)}. \tag{3.2.20}$$

So substituting (3.2.20) into (3.2.19) we get the recursive formula (3.2.15).

2. The formula (3.2.16) for $p = L$ follows directly from the way we defined $\theta_{tkl}^{(L)}$, $\psi_{tkl}^{(L)}$, and equation (3.2.8). Next, define the $\theta_{tjb}^{(p)}$ as above, and set

$$\psi_{tjb}^{(p)} = \left[\sum_{\eta=1}^{K_{p+1}} \frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}} \overline{w_{\eta j}^{(p)}}\right] g_p'\left(\overline{\left(x_{tj}^{(p)}\right)^{\text{net}}}\right) g_p'\left(\overline{\left(x_{tb}^{(p)}\right)^{\text{net}}}\right) \tag{3.2.21}$$

in (3.2.11). Substituting (3.2.21) and (3.2.17) into (3.2.11) gives us (3.2.16). For the $\psi_{tjb}^{(p)}$, using (3.2.12) with our definition of the $\psi_{tjb}^{(p)}$ in (3.2.21) we have:

$$\frac{\partial E_{t\eta}^{(p+1)}}{\partial \overline{x_{tb}^{(p)}}} = \sum_{\beta=1}^{K_{p+1}} \left(\psi_{t\eta\beta}^{(p+1)} \overline{w_{\beta b}^{(p)}} + \theta_{t\eta\beta}^{(p+1)} \overline{w_{\eta b}^{(p)}}\right) \tag{3.2.22}$$

so substituting (3.2.22) into (3.2.21) we get (3.2.18).

□

3.3 Backpropagation Using the Pseudo-Newton Method

To simplify the computation in the implementation of Newton's method, we can use the pseudo-Newton algorithm, which is an alternative algorithm also known to

provide good quadratic convergence. For the pseudo-Newton algorithm, we take $\mathcal{H}_{\overline{\mathbf{w}^{(p-1)}}\mathbf{w}^{(p-1)}} = 0 = \mathcal{H}_{\mathbf{w}^{(p-1)}\overline{\mathbf{w}^{(p-1)}}}$ in (3.2.13), thus reducing the weight updates to

$$\Delta\mathbf{w}^{(p-1)} = -\mathcal{H}^{-1}_{\mathbf{w}^{(p-1)}\mathbf{w}^{(p-1)}} \left(\frac{\partial E}{\partial \mathbf{w}^{(p-1)}}\right)^*.$$

Convergence using the pseudo-Newton algorithm will generally be faster than gradient descent. The trade off for computational efficiency over Newton's method is somewhat slower convergence, though if the activation functions in the holomorphic MLP are in addition onto, the performance of the pseudo-Newton versus Newton algorithms should be similar [29].

Corollary 3.3.1 (Pseudo-Newton Backpropagation Algorithm for Holomorphic Neural Networks, [30], Corollary 5.1)**.** *The weight updates for the holomorphic MLP with activation functions satisfying*

$$\overline{g(z)} = g(\overline{z}),$$

$p = 1, ..., L$*, using the backpropagation algorithm with the pseudo-Newton's method are given by*

$$\Delta\mathbf{w}^{(p-1)} = -\mathcal{H}^{-1}_{\mathbf{w}^{(p-1)}\mathbf{w}^{(p-1)}} \left(\frac{\partial E}{\partial \mathbf{w}^{(p-1)}}\right)^*,$$

where the entries of the Hessian matrices $\mathcal{H}_{\mathbf{w}^{(p-1)}\mathbf{w}^{(p-1)}}$ *for* $1 \leq p \leq L$ *are given by (3.2.14) in Theorem 3.2.1.*

3.4 The One-Step Newton Steplength Algorithm for Real-Valued Complex Functions

A significant problem encountered with Newton's method and other minimization algorithms is the tendency of the iterates to "overshoot." If this happens, the iterates may not decrease the function value at each step [38]. For functions on real domains, it is known that for any minimization algorithm, careful choice of the sequence of steplengths via various steplength algorithms will guarantee a descent method. Steplength algorithms for minimization of real-valued functions on complex

domains have been discussed in the literature [7,21,35,47]. In [35], the problem was addressed by imposing unitary conditions on the input vectors. In [47], steplength algorithms were proposed for the BFGS method, which is an approximation to Newton's method. With regard to applications in neural networks, variable steplength algorithms exist for least mean square error algorithms, and these algorithms have been adapted to the gradient descent backpropagation algorithm for fully complex-valued neural networks with analytic activation functions [7, 18]. Fully adaptive gradient descent algorithms for complex-valued neural networks have also been proposed [21]. However, these algorithms do not apply to the Newton backpropagation algorithm.

We provide a steplength algorithm that guarantees convergence of Newton's method for real-valued complex functions. Let $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$, and consider a general minimization algorithm with sequence of iterates $\{\mathbf{z}(n)\}$ given recursively by

$$\mathbf{z}(n+1) = \mathbf{z}(n) - \mu(n)\mathbf{p}(n),\ n = 0, 1, ..., \tag{3.4.1}$$

where $\mathbf{p}(n) \in \mathbb{C}^k$ such that $-\mathbf{p}(n)$ is the direction from the nth iterate to the $(n+1)$th iterate and $\mu(n) \in \mathbb{R}$ is the learning rate or steplength which we allow to vary with each step. We are interested in guaranteeing that the minimization algorithm is a descent method, that is, that at each stage of the iteration the inequality $f(\mathbf{z}(n+1)) \leq f(\mathbf{z}(n))$ for $n = 0, 1, ...$ holds. Here, we provide details of the proof of the one-step Newton steplength algorithm for the minimization of real-valued functions on complex domains. Our treatment follows the exposition in [38] with notation employed from [29], with the application to the complex Newton algorithm providing a proof of Theorem 3.4.4.

Lemma 3.4.1. *Suppose that $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ is $\mathbb{R}$-differentiable at $\mathbf{z} \in \text{int}(\Omega)$ and that there exists $\mathbf{p} \in \mathbb{C}^k$ such that $\text{Re}\left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z})\mathbf{p}\right) > 0$. Then there exists a $\delta > 0$ such that $f(\mathbf{z} - \mu\mathbf{p}) < f(\mathbf{z})$ for all $\mu \in (0, \delta)$.*

Proof. Let $\mathbf{z} = \mathbf{x} + i\mathbf{y} \in \text{int}(\Omega)$ with $\mathbf{x}, \mathbf{y} \in \mathbb{R}^k$. The function $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ is $\mathbb{R}$-differentiable at $\mathbf{z}$ if and only if $f : D \subseteq \mathbb{R}^{2k} \to \mathbb{R}$ is (Frechet) differentiable at $(\mathbf{x}, \mathbf{y})^T \in \text{int}(D)$, where D is defined as in (3.1.3) and the (Frechet) derivative (equal to the Gateau derivative) at $(\mathbf{x}, \mathbf{y})^T$ is given by $\left(\frac{\partial f}{\partial \mathbf{x}}, \frac{\partial f}{\partial \mathbf{y}}\right)$. Suppose there exists $\mathbf{p} = \mathbf{p}_R + i\mathbf{p}_I \in \mathbb{C}^k$ with $\mathbf{p}_R, \mathbf{p}_I \in \mathbb{R}^k$ such that $\text{Re}\left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z})\mathbf{p}\right) > 0$. Then using the coordinate and cogradient transformations (3.1.1) and (3.1.2) and the fact that

f is real-valued, we have the following ([29], pg. 34):

$$\left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x},\mathbf{y}), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x},\mathbf{y})\right)\begin{pmatrix}\mathbf{p}_R\\ \mathbf{p}_I\end{pmatrix} = \left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z},\overline{\mathbf{z}}), \frac{\partial f}{\partial \overline{\mathbf{z}}}(\mathbf{z},\overline{\mathbf{z}})\right) J \cdot \frac{1}{2} J^* \begin{pmatrix}\mathbf{p}\\ \overline{\mathbf{p}}\end{pmatrix}$$
$$= \frac{\partial f}{\partial \mathbf{z}}(\mathbf{z},\overline{\mathbf{z}})\mathbf{p} + \frac{\partial f}{\partial \overline{\mathbf{z}}}(\mathbf{z},\overline{\mathbf{z}})\overline{\mathbf{p}} = \frac{\partial f}{\partial \mathbf{z}}(\mathbf{z})\mathbf{p} + \overline{\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z})\mathbf{p}} = 2\mathrm{Re}\left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z})\mathbf{p}\right) > 0. \tag{3.4.2}$$

By (8.2.1) in [38] there exists a $\delta > 0$ such that $f((\mathbf{x},\mathbf{y}) - \mu(\mathbf{p}_R, \mathbf{p}_I)) < f(\mathbf{x},\mathbf{y})$ for all $\mu \in (0,\delta)$. Viewing f again as a function on the complex domain Ω, this is equivalent to the statement that $f(\mathbf{z} - \mu\mathbf{p}) < f(\mathbf{z})$ for all $\mu \in (0,\delta)$. □

We define a stationary point of f to be a stationary point in the sense of the function $f(\mathbf{z}) = f(\mathbf{x},\mathbf{y}) : D \subseteq \mathbb{R}^{2k} \to \mathbb{R}$. If $\hat{\mathbf{z}} = \hat{\mathbf{x}} + i\hat{\mathbf{y}}$ with $\hat{\mathbf{x}}, \hat{\mathbf{y}} \in \mathbb{R}^k$, then $\hat{\mathbf{z}}$ is a stationary point of f if and only if $\frac{\partial f}{\partial \mathbf{x}}(\hat{\mathbf{x}},\hat{\mathbf{y}}) = \frac{\partial f}{\partial \mathbf{y}}(\hat{\mathbf{x}},\hat{\mathbf{y}}) = 0$. Note that if $\mathrm{Re}\left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z})\right) \neq 0$ for $\mathbf{z} \in \mathrm{int}(\Omega)$ (i.e. $\mathbf{z}$ is not a stationary point), then there always exists a $\mathbf{p} \in \mathbb{C}^k$ such that $\mathrm{Re}\left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z})\mathbf{p}\right) > 0$. So this result is always true in the real domain, and the proof of Lemma 3.4.1 only translates the result from the real domain to the complex domain.

For the sequence of iterates $\{\mathbf{z}(n)\}$ given by (3.4.1), we can find a sequence $\{\mathbf{p}(n)\}$ such that $\mathrm{Re}\left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n))\mathbf{p}(n)\right) > 0$ for $n = 0, 1, \ldots$. By Lemma 3.4.1, for each n there is at least one $\mu(n) \in (0,\infty)$ such that $f(\mathbf{z}(n) - \mu(n)\mathbf{p}(n)) < f(\mathbf{z}(n))$. At each step in the algorithm we would like to make the largest descent in the value of f as possible, so finding a desirable steplength $\mu(n)$ to guarantee descent translates into the real one-dimensional problem of minimizing $f(\mathbf{z}(n) - \mu\mathbf{p}(n))$ as a function of μ. For each n let $\mathbf{z}(n) = \mathbf{x}(n) + i\mathbf{y}(n)$ and $\mathbf{p}(n) = \mathbf{p}_R(n) + i\mathbf{p}_I(n)$ with $\mathbf{x}(n), \mathbf{y}(n), \mathbf{p}_R(n), \mathbf{p}_I(n) \in \mathbb{R}^k$ and write

$$f(\mathbf{z}(n) - \mu\mathbf{p}(n)) = f((\mathbf{x}(n),\mathbf{y}(n)) - \mu(\mathbf{p}_R(n), \mathbf{p}_I(n))).$$

Suppose f is twice $\mathbb{R}$-differentiable on Ω. As an approximate solution to this one-dimensional minimization problem we take $\mu(n)$ to be the minimizer of the second-

degree Taylor polynomial (in μ)

$$T_2(\mu) = f(\mathbf{x}(n), \mathbf{y}(n)) - \mu \left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}(n), \mathbf{y}(n)), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x}(n), \mathbf{y}(n)) \right) \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix} + \frac{1}{2}\mu^2 \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}^T \mathcal{H}_{\mathbf{rr}}(\mathbf{x}(n), \mathbf{y}(n)) \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix} \tag{3.4.3}$$

where $\mathcal{H}_{\mathbf{rr}}$ denotes the real Hessian matrix

$$\mathcal{H}_{\mathbf{rr}} = \left(\frac{\partial}{\partial \mathbf{x}}, \frac{\partial}{\partial \mathbf{y}} \right) \left(\frac{\partial f}{\partial \mathbf{x}}, \frac{\partial f}{\partial \mathbf{y}} \right)^T.$$

If

$$\begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}^T \mathcal{H}_{\mathbf{rr}}(\mathbf{x}(n), \mathbf{y}(n)) \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix} > 0$$

then T_2 has a minimum at

$$\mu(n) = \frac{\left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}(n), \mathbf{y}(n)), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x}(n), \mathbf{y}(n)) \right) \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}}{\begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}^T \mathcal{H}_{\mathbf{rr}}(\mathbf{x}(n), \mathbf{y}(n)) \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}}. \tag{3.4.4}$$

(Note this is equivalent to taking one step toward minimizing f over μ via the real Newton algorithm.) Using a computation similar to (3.4.2) in the proof of Lemma 3.4.1, the denominator of (3.4.4) translates back into complex coordinates as ([29], pg. 38):

$$\begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}^T \mathcal{H}_{\mathbf{rr}}(\mathbf{x}(n), \mathbf{y}(n)) \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix} = 2\mathrm{Re}\left\{ \mathbf{p}(n)^* \mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n)) \mathbf{p}(n) + \mathbf{p}(n)^* \mathcal{H}_{\bar{\mathbf{z}}\mathbf{z}}(\mathbf{z}(n)) \overline{\mathbf{p}(n)} \right\}. \tag{3.4.5}$$

Combining (3.4.4) with (3.4.5) and (3.4.2), if

$$\mathrm{Re}\left\{\mathbf{p}(n)^*\mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n))\mathbf{p}(n) + \mathbf{p}(n)^*\mathcal{H}_{\bar{\mathbf{z}}\mathbf{z}}(\mathbf{z}(n))\overline{\mathbf{p}(n)}\right\} > 0 \tag{3.4.6}$$

we can take the approximate solution to the minimization problem to be

$$\mu(n) = \frac{\mathrm{Re}\left\{\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n))\mathbf{p}(n)\right\}}{\mathrm{Re}\left\{\mathbf{p}(n)^*\mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n))\mathbf{p}(n) + \mathbf{p}(n)^*\mathcal{H}_{\bar{\mathbf{z}}\mathbf{z}}(\mathbf{z}(n))\overline{\mathbf{p}(n)}\right\}}. \tag{3.4.7}$$

Notice that (3.4.6) is in fact both a necessary and sufficient condition to obtain an approximate solution using (3.4.3) to the one-dimensional minimization problem of $f(\mathbf{z}(n) - \mu\mathbf{p}(n))$ over μ, for if

$$\mathrm{Re}\left\{\mathbf{p}(n)^*\mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n))\mathbf{p}(n) + \mathbf{p}(n)^*\mathcal{H}_{\bar{\mathbf{z}}\mathbf{z}}(\mathbf{z}(n))\overline{\mathbf{p}(n)}\right\} < 0,$$

the Taylor polynomial (3.4.3) attains only a maximum.

Since defining the sequence of steplengths $\{\mu(n)\}$ by (3.4.7) is only an approximate method, to guarantee the descent of the iteration, we consider further modification of the steplengths. From Lemma 3.4.1, it is clear that we can choose a sequence of underrelaxation factors $\{\omega(n)\}$ such that

$$f(\mathbf{z}(n) - \omega(n)\mu(n)\mathbf{p}(n)) < f(\mathbf{z}(n))$$

which guarantees that the iteration

$$\mathbf{z}(n+1) = \mathbf{z}(n) - \omega(n)\mu(n)\mathbf{p}(n),\ n = 0, 1, ... \tag{3.4.8}$$

is a descent method. We describe a way to choose the sequence $\{\omega(n)\}$.

Suppose Ω is open and let $\mathbf{z}(0) \in \Omega$. The level set of $\mathbf{z}(0)$ under f on Ω is defined by

$$L_{\mathbb{C}^k}(f(\mathbf{z}(0))) = \{\mathbf{z} \in \Omega \,|\, f(\mathbf{z}) \leq f(\mathbf{z}(0))\}, \tag{3.4.9}$$

and $L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$ is the path-connected component of $L_{\mathbb{C}^k}(f(\mathbf{z}(0))$ containing $\mathbf{z}(0)$. Let $\|\cdot\|_{\mathbb{C}^k} : \mathbb{C}^k \to \mathbb{R}$ denote the Euclidean norm on $\mathbb{C}^k$, with $\|\mathbf{z}\|_{\mathbb{C}^k} = \sqrt{\mathbf{z}^*\mathbf{z}}$.

Lemma 3.4.2 (Complex Version of the One-Step Newton Steplength Algorithm)**.** *Let $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ be twice-continuously $\mathbb{R}$-differentiable on the open set Ω. Suppose $L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$ is compact for $\mathbf{z}(0) \in \Omega$ and that*

$$\eta_0 \mathbf{h}^* \mathbf{h} \leq \mathrm{Re}\{\mathbf{h}^* \mathcal{H}_{\mathbf{zz}}(\mathbf{z})\mathbf{h} + \mathbf{h}^* \mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z})\overline{\mathbf{h}}\} \leq \eta_1 \mathbf{h}^* \mathbf{h} \tag{3.4.10}$$

for all $\mathbf{z} \in L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$ and $\mathbf{h} \in \mathbb{C}^k$, where $0 < \eta_0 \leq \eta_1$. Fix $\epsilon \in (0, 1]$. Define the sequence $\{\mathbf{z}(n)\}$ using (3.4.8) with $\mathbf{p}(n) \neq 0$ satisfying

$$\mathrm{Re}\left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n))(\mathbf{p}(n))\right) \geq 0, \tag{3.4.11}$$

$\mu(n)$ defined by (3.4.7), and

$$0 < \epsilon \leq \omega(n) \leq \frac{2}{\gamma(n)} - \epsilon, \tag{3.4.12}$$

where, setting $\mathbf{z} = \mathbf{z}(n)$ and $\mathbf{p} = \mathbf{p}(n)$,

$$\gamma(n) = \sup\left\{ \frac{\mathrm{Re}\{\mathbf{p}^* \mathcal{H}_{\mathbf{zz}}(\mathbf{z} - \mu\mathbf{p})\mathbf{p} + \mathbf{p}^* \mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z} - \mu\mathbf{p})\overline{\mathbf{p}}\}}{\mathrm{Re}\{\mathbf{p}^* \mathcal{H}_{\mathbf{zz}}(\mathbf{z})\mathbf{p} + \mathbf{p}^* \mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z})\overline{\mathbf{p}}\}} \,\middle|\, \begin{array}{c} \mu > 0,\ f(\mathbf{z} - \nu\mathbf{p}) < f(\mathbf{z}) \\ \textit{for all } \nu \in (0, \mu] \end{array} \right\}. \tag{3.4.13}$$

Then $\{\mathbf{z}(n)\} \subseteq L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$,

$$\lim_{n \to \infty} \frac{\mathrm{Re}\left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n))(\mathbf{p}(n))\right)}{\|\mathbf{p}(n)\|_{\mathbb{C}^k}} = 0,$$

and $\lim_{n\to\infty}(\mathbf{z}(n) - \mathbf{z}(n+1)) = 0$.

Proof. Let $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ be twice-continuously $\mathbb{R}$-differentiable on the open set Ω, and define D as in (3.1.3). Then D is open and $f(\mathbf{x}, \mathbf{y}) : D \subseteq \mathbb{R}^{2k} \to \mathbb{R}$ is twice-continuously differentiable on D. Let $\mathbf{z}(0) = \mathbf{x}(0) + i\mathbf{y}(0) \in \Omega$ with $\mathbf{x}(0), \mathbf{y}(0) \in \mathbb{R}^k$

and set

$$L^0_{\mathbb{R}^{2k}}(f(\mathbf{x}(0),\mathbf{y}(0)) = \left\{ \left(\begin{array}{c} \mathbf{x} \\ \mathbf{y} \end{array} \right) \in D \;\middle|\; \begin{array}{c} \mathbf{x},\mathbf{y} \in \mathbb{R}^k, \\ \mathbf{z} = \mathbf{x} + i\mathbf{y} \in L^0_{\mathbb{C}^k}(f(\mathbf{z}(0))) \end{array} \right\}.$$

It is clear that since $L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$ is assumed to be compact, the real level set $L^0_{\mathbb{R}^{2k}}(f(\mathbf{x}(0),\mathbf{y}(0))$ is also compact.

Next, observe that for $\mathbf{z} = \mathbf{x} + i\mathbf{y} \in \mathbb{C}^k$ with $\mathbf{x},\mathbf{y} \in \mathbb{R}^k$, if $\|\cdot\|_{R^{2k}} : \mathbb{R}^{2k} \to \mathbb{R}$ denotes the Euclidean norm on $\mathbb{R}^{2k}$, then

$$\|\mathbf{z}\|^2_{\mathbb{C}^k} = \mathbf{z}^*\mathbf{z} = \left\| \left(\begin{array}{c} \mathbf{x} \\ \mathbf{y} \end{array} \right) \right\|^2_{\mathbb{R}^{2k}}.$$

Using this fact and (3.4.5) we see that for $\mathbf{z} = \mathbf{x} + i\mathbf{y} \in L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$ and $\mathbf{h} = \mathbf{h}_R + i\mathbf{h}_I \in \mathbb{C}^k$ with $\mathbf{x},\mathbf{y},\mathbf{h}_R,\mathbf{h}_I \in \mathbb{R}^k$ the condition (3.4.10) is equivalent to

$$\eta'_0 \left\| \left(\begin{array}{c} \mathbf{h}_R \\ \mathbf{h}_I \end{array} \right) \right\|^2_{\mathbb{R}^{2k}} \leq \left(\begin{array}{c} \mathbf{h}_R \\ \mathbf{h}_I \end{array} \right)^T \mathcal{H}_{\mathbf{rr}}(\mathbf{x},\mathbf{y}) \left(\begin{array}{c} \mathbf{h}_R \\ \mathbf{h}_I \end{array} \right) \leq \eta'_1 \left\| \left(\begin{array}{c} \mathbf{h}_R \\ \mathbf{h}_I \end{array} \right) \right\|^2_{\mathbb{R}^{2k}},$$

where again $\mathcal{H}_{\mathbf{rr}}$ denotes the real Hessian matrix of $f(\mathbf{x},\mathbf{y}) : D \subseteq \mathbb{R}^{2k} \to \mathbb{R}$, and $0 < \eta'_0 = \frac{\eta_0}{2} \leq \frac{\eta_1}{2} = \eta'_1$.

We have already seen in the proof of Lemma 3.4.1 (see the calculation (3.4.2)) that the condition (3.4.11) on the vectors $\mathbf{p}(n) = \mathbf{p}_R(n) + i\mathbf{p}_I(n)$ with $\mathbf{p}_R(n),\mathbf{p}_I(n) \in \mathbb{R}^k$ is equivalent to the real condition

$$\left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}(n),\mathbf{y}(n)), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x}(n),\mathbf{y}(n)) \right) \left(\begin{array}{c} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{array} \right) \geq 0.$$

We have also seen that our choice (3.4.7) for $\mu(n)$ is equal to (3.4.4).

Finally, for $\epsilon \in (0,1]$, using (3.4.5) again we have the real analogue of (3.4.13):

$$\gamma(n) = \sup \left\{ \left| \frac{\begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}^T \mathcal{H}_{\mathbf{rr}}((\mathbf{x}(n),\mathbf{y}(n)) - \mu(\mathbf{p}_R(n),\mathbf{p}_I(n))) \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}}{\begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}^T \mathcal{H}_{\mathbf{rr}}(\mathbf{x}(n),\mathbf{y}(n)) \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}} \right| \right.$$
$$\left. \mu > 0,\ f((\mathbf{x}(n),\mathbf{y}(n)) - \nu(\mathbf{p}_R(n),\mathbf{p}_I(n))) < f(\mathbf{x}(n),\mathbf{y}(n)) \text{ for all } \nu \in (0,\mu] \right\}.$$

By (3.4.2),

$$\frac{\left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}(n),\mathbf{y}(n)), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x}(n),\mathbf{y}(n))\right) \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}}{\left\| \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix} \right\|_{\mathbb{R}^{2k}}} = \frac{2\mathrm{Re}\left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n))\mathbf{p}(n)\right)}{\|\mathbf{p}(n)\|_{\mathbb{C}^k}},$$

so applying (14.2.9) in [38], $\left\{(\mathbf{x}(n),\mathbf{y}(n))^T\right\} \subseteq L^0_{\mathbb{R}^{2k}}(f(\mathbf{x}(0),\mathbf{y}(0)))$,

$$\lim_{n\to\infty} \frac{\left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}(n),\mathbf{y}(n)), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x}(n),\mathbf{y}(n))\right) \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix}}{\left\| \begin{pmatrix} \mathbf{p}_R(n) \\ \mathbf{p}_I(n) \end{pmatrix} \right\|_{\mathbb{R}^{2k}}} = 0,$$

and

$$\lim_{n\to\infty} \left(\begin{pmatrix} \mathbf{x}(n) \\ \mathbf{y}(n) \end{pmatrix} - \begin{pmatrix} \mathbf{x}(n+1) \\ \mathbf{y}(n+1) \end{pmatrix} \right) = 0.$$

Translating back to complex coordinates yields the desired conclusion. □

Assume that there is a unique stationary point $\hat{\mathbf{z}}$ in $L^0_{\mathbb{C}^k}(f(\mathbf{z}(0))$. We desire to

guarantee that the sequence of iterates $\{\mathbf{z}(n)\}$ converges to $\hat{\mathbf{z}}$. Before we give conditions for convergence of the complex version of the one-step Newton steplength algorithm, recall that the root-convergence factors (R-factors) of a sequence $\{\mathbf{z}(n)\} \subseteq \mathbb{C}^k$ that converges to $\hat{\mathbf{z}} \in \mathbb{C}^k$ are

$$R_p\{\mathbf{z}(n)\} = \begin{cases} \limsup_{n\to\infty} \|\mathbf{z}(n) - \hat{\mathbf{z}}\|_{\mathbb{C}^k}^{1/n} & \text{if } p = 1, \\ \limsup_{n\to\infty} \|\mathbf{z}(n) - \hat{\mathbf{z}}\|_{\mathbb{C}^k}^{1/p^n} & \text{if } p > 1, \end{cases} \tag{3.4.14}$$

and the sequence is said to have at least an R-linear rate of convergence if $R_1\{\mathbf{z}(n)\} < 1$.

Lemma 3.4.3 (Convergence of the Complex Version of the One-Step Newton Steplength Algorithm)**.** *Let $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ be twice-continuously $\mathbb{R}$-differentiable on the open convex set Ω and assume that $L^0_{\mathbb{C}^k}(f(\mathbf{z}(0))$ is compact for $\mathbf{z}(0) \in \Omega$. Assume the notation as in Lemma 3.4.2. Suppose for all $z \in \Omega$,*

$$\mathrm{Re}\{\mathbf{h}^* \mathcal{H}_{\mathbf{zz}}(\mathbf{z})\mathbf{h} + \mathbf{h}^* \mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z})\overline{\mathbf{h}}\} > 0 \text{ for all } \mathbf{h} \in \mathbb{C}^k, \tag{3.4.15}$$

and assume that the $\mathbf{p}(n)$ are nonzero vectors satisfying

$$\mathrm{Re}\left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n))\mathbf{p}(n)\right) \geq C \left\| \left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n))\right)^T \right\|_{\mathbb{C}^k} \|\mathbf{p}(n)\|_{\mathbb{C}^k}, \ n = 0, 1, ... \tag{3.4.16}$$

for some fixed $C > 0$. Assume f has a unique stationary point $\hat{\mathbf{z}}$ in $L^0_{\mathbb{C}^k}(f(\mathbf{z}(0))$. Then $\lim_{n\to\infty} \mathbf{z}(n) = \hat{\mathbf{z}}$, and the rate of convergence is at least R-linear.

Proof. As in the proof of Lemma 3.4.2, given the assumptions of this lemma, $f : D \subseteq \mathbb{R}^{2k} \to \mathbb{R}$ is twice-continuously (Frechet) differentiable on the open convex set D, and the set $L^0_{\mathbb{R}^{2k}}(f(\mathbf{x}(0), \mathbf{y}(0))$ is compact for $\mathbf{z}(0) = \mathbf{x}(0) + i\mathbf{y}(0) \in \Omega$, where $\mathbf{x}(0), \mathbf{y}(0) \in \mathbb{R}^k$.

Using (3.4.5), for $\mathbf{z} = \mathbf{x} + i\mathbf{y} \in \Omega$ the condition (3.4.15) is equivalent to the condition

$$\begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{pmatrix}^T \mathcal{H}_{\mathbf{rr}}(\mathbf{x}, \mathbf{y}) \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{pmatrix} > 0 \text{ for all } \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{pmatrix} \in \mathbb{R}^{2k} \text{ with } \mathbf{h}_1, \mathbf{h}_2 \in \mathbb{R}^k.$$

Thus for all $(\mathbf{x},\mathbf{y})^T \in D$, the real Hessian $\mathcal{H}_{\mathbf{rr}}(\mathbf{x},\mathbf{y})$ of f is positive definite.

Also as in the proof of Lemma 3.4.2, the real versions of the definitions of $\mu(n)$ and $\omega(n)$ given by (3.4.7) and (3.4.12), respectively, satisfy the real one-step Newton steplength algorithm (14.2.9) in [38].

Finally, for $\mathbf{z} = \mathbf{x} + i\mathbf{y} \in \mathbb{C}^k$ with $\mathbf{x},\mathbf{y} \in \mathbb{R}^k$, a simple calculation shows that

$$2\left\|\left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z})\right)^T\right\|_{\mathbb{C}^k} = \left\|\left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x},\mathbf{y}), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x},\mathbf{y})\right)^T\right\|_{\mathbb{R}^{2k}},$$

so using the calculation (3.4.2) in the proof of Lemma 3.4.1, the condition (3.4.16) for the nonzero vectors $\mathbf{p}(n) = \mathbf{p}_R(n) + i\mathbf{p}_I(n)$ with $\mathbf{p}_R(n), \mathbf{p}_I(n) \in \mathbb{R}^k$ is equivalent to the real condition

$$\left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x},\mathbf{y}), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x},\mathbf{y})\right)\begin{pmatrix}\mathbf{p}_R(n)\\ \mathbf{p}_I(n)\end{pmatrix} \geq C\left\|\left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x},\mathbf{y}), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x},\mathbf{y})\right)^T\right\|_{\mathbb{R}^{2k}}\left\|\begin{pmatrix}\mathbf{p}_R(n)\\ \mathbf{p}_I(n)\end{pmatrix}\right\|_{\mathbb{R}^{2k}}.$$

Thus we may apply Theorem (14.3.6) in [38] and transfer back to complex coordinates to obtain that $\lim_{n\to\infty} \mathbf{z}(n) = \hat{\mathbf{z}}$, where $\hat{\mathbf{z}} = \hat{\mathbf{x}} + i\hat{\mathbf{y}}$ with $\hat{\mathbf{x}},\hat{\mathbf{y}} \in \mathbb{R}^k$ is the unique stationary point of f in $L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$, and the rate of convergence is at least R-linear. □

The following theorem gives the one-step Newton steplength algorithm to adjust the sequence of steplengths for minimization of a real-valued complex function using Newton's method.

Theorem 3.4.4 (Convergence of the Complex Newton Algorithm with Complex One-Step Newton Steplengths, [30], Theorem 6.1)**.** *Let $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ be twice-continuously $\mathbb{R}$-differentiable on the open convex set Ω and assume that $L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$ is compact for $\mathbf{z}(0) \in \Omega$. Suppose for all $\mathbf{z} \in \Omega$,*

$$\mathrm{Re}\{\mathbf{h}^*\mathcal{H}_{\mathbf{zz}}(\mathbf{z})\mathbf{h} + \mathbf{h}^*\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z})\overline{\mathbf{h}}\} > 0 \text{ for all } \mathbf{h} \in \mathbb{C}^k.$$

Assume f has a unique stationary point $\hat{\mathbf{z}} \in L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$, and fix $\epsilon \in (0,1]$. Con-

sider the iteration

$$\mathbf{z}(n+1) = \mathbf{z}(n) - \omega(n)\mu(n)\mathbf{p}(n), \ n = 0, 1, ..., \tag{3.4.17}$$

where the $\mathbf{p}(n)$ *are the nonzero complex Newton updates*

$$\begin{aligned}\mathbf{p}(\mathbf{z}(n)) = &- \left[\mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n)) - \mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z}(n))\mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}(\mathbf{z}(n))^{-1}\mathcal{H}_{\mathbf{z}\overline{\mathbf{z}}}(\mathbf{z}(n))\right]^{-1} \\ &\cdot \left[\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z}(n))\mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}(\mathbf{z}(n))^{-1}\left(\frac{\partial f}{\partial \overline{\mathbf{z}}}(\mathbf{z}(n))\right)^{*} - \left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n))\right)^{*}\right],\end{aligned} \tag{3.4.18}$$

the steplengths $\mu(n)$ *are given by*

$$\mu(n) = \frac{\mathrm{Re}\{\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n))\mathbf{p}(n)\}}{\mathrm{Re}\{\mathbf{p}(n)^{*}\mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n))\mathbf{p}(n) + \mathbf{p}(n)^{*}\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z}(n))\overline{\mathbf{p}(n)}\}},$$

and the underrelaxation factors $\omega(n)$ *satisfy*

$$0 \leq \epsilon \leq \omega(n) \leq \frac{2}{\gamma(n)} - \epsilon, \tag{3.4.19}$$

where, taking $\mathbf{z} = \mathbf{z}(n)$ *and* $\mathbf{p} = \mathbf{p}(n)$,

$$\begin{aligned}&\gamma(n) \\ &= \sup\left\{\frac{\mathrm{Re}\{\mathbf{p}^{*}\mathcal{H}_{\mathbf{zz}}(\mathbf{z}-\mu\mathbf{p})\mathbf{p} + \mathbf{p}^{*}\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z}-\mu\mathbf{p})\overline{\mathbf{p}}\}}{\mathrm{Re}\{\mathbf{p}^{*}\mathcal{H}_{\mathbf{zz}}(\mathbf{z})\mathbf{p} + \mathbf{p}^{*}\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z})\overline{\mathbf{p}}\}}\middle| \begin{array}{c}\mu > 0, \ f(\mathbf{z}-\nu\mathbf{p}) < f(\mathbf{z}) \\ \textit{for all } \nu \in (0, \mu]\end{array}\right\}.\end{aligned} \tag{3.4.20}$$

Then $\lim_{n\to\infty}\mathbf{z}(n) = \hat{\mathbf{z}}$, *and the rate of convergence is at least R-linear.*

Proof. We apply the previous results to the complex Newton algorithm. Let $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ be twice-continuously $\mathbb{R}$-differentiable on the open convex set Ω. Let $\mathbf{z}(0) \in \Omega$ and assume that the level set $L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$ is compact. Suppose for all

$\mathbf{z} \in \Omega$,

$$\mathrm{Re}\{\mathbf{h}^* \mathcal{H}_{\mathbf{zz}}(\mathbf{z})\mathbf{h} + \mathbf{h}^* \mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z})\overline{\mathbf{h}}\} > 0 \text{ for all } \mathbf{h} \in \mathbb{C}^k.$$

As in the proof of Lemma 3.4.3, this condition is equivalent to the positive definiteness of the real Hessian matrix $\mathcal{H}_{\mathbf{rr}}(\mathbf{x}, \mathbf{y})$ of f for all $(\mathbf{x}, \mathbf{y})^T \in D$. Since f is twice-continuously $\mathbb{R}$-differentiable, the Hessian operator $\mathcal{H}_{\mathbf{rr}}(\cdot) : D \subseteq \mathbb{R}^{2k} \to L(\mathbb{R}^{2k})$ (where $L(\mathbb{R}^{2k})$ denotes the set of linear operators $\mathbb{R}^{2k} \to \mathbb{R}^{2k}$) is continuous. Restricting to the compact set $L^0_{\mathbb{R}^{2k}}(f(\mathbf{x}(0), \mathbf{y}(0)))$ we have that $\mathcal{H}_{\mathbf{rr}}(\cdot) : L^0_{\mathbb{R}^{2k}}(f(\mathbf{x}(0), \mathbf{y}(0))) \to L(\mathbb{R}^{2k})$ is a continuous mapping such that $\mathcal{H}_{\mathbf{rr}}(\mathbf{x}, \mathbf{y})$ is positive definite for each vector $(\mathbf{x}, \mathbf{y})^T \in L^0_{\mathbb{R}^{2k}}(f(\mathbf{x}(0), \mathbf{y}(0)))$. For each $(\mathbf{x}, \mathbf{y})^T \in L^0_{\mathbb{R}^{2k}}(f(\mathbf{x}(0), \mathbf{y}(0)))$ set

$$\tilde{\mathbf{p}}(\mathbf{x}, \mathbf{y}) = \mathcal{H}_{\mathbf{rr}}(\mathbf{x}, \mathbf{y})^{-1} \left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}, \mathbf{y}), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x}, \mathbf{y}) \right)^T.$$

By Lemma (14.4.1) in [38], there exists a constant $C > 0$ such that

$$\left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}, \mathbf{y}), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x}, \mathbf{y}) \right) \tilde{\mathbf{p}}(\mathbf{x}, \mathbf{y}) \geq C \left\| \left(\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}, \mathbf{y}), \frac{\partial f}{\partial \mathbf{y}}(\mathbf{x}, \mathbf{y}) \right)^T \right\|_{\mathbb{R}^{2k}} \|\tilde{\mathbf{p}}(\mathbf{x}, \mathbf{y})\|_{\mathbb{R}^{2k}} \tag{3.4.21}$$

for all $(\mathbf{x}, \mathbf{y})^T \in L^0_{\mathbb{R}^{2k}}(f(\mathbf{x}(0), \mathbf{y}(0)))$. As in the proof of Lemma 3.4.3, (3.4.21) is equivalent to the inequality

$$\mathrm{Re} \left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}) \mathbf{p}(\mathbf{z}) \right) \geq C \left\| \left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}) \right)^T \right\|_{\mathbb{C}^k} \|\mathbf{p}(\mathbf{z})\|_{\mathbb{C}^k}$$

for all $\mathbf{z} \in L^0_{\mathbb{R}^{2k}}(f(\mathbf{x}(0), \mathbf{y}(0)))$, where

$$\begin{aligned} \tilde{\mathbf{p}}(\mathbf{z}) = &- \left[\mathcal{H}_{\mathbf{zz}}(\mathbf{z}) - \mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z}) \mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}(\mathbf{z})^{-1} \mathcal{H}_{\mathbf{z}\overline{\mathbf{z}}}(\mathbf{z}) \right]^{-1} \\ &\cdot \left[\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z}) \mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}(\mathbf{z})^{-1} \left(\frac{\partial f}{\partial \overline{\mathbf{z}}}(\mathbf{z}) \right)^* - \left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}) \right)^* \right] \end{aligned}$$

is obtained from $\tilde{\mathbf{p}}(\mathbf{x}, \mathbf{y})$ (where $\mathbf{z} = \mathbf{x} + i\mathbf{y}$) using the coordinate and cogradient transformations (3.1.1) and (3.1.2), respectively [29]. Suppose f has a unique

stationary point $\hat{\mathbf{z}}$ in $L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$. Consider the iteration

$$\mathbf{z}(n+1) = \mathbf{z}(n) - \omega(n)\mu(n)\mathbf{p}(n),\ n = 0, 1, ...,$$

where the $\mathbf{p}(n)$ are the nonzero complex Newton updates defined by $\mathbf{p}(n) = \tilde{\mathbf{p}}(\mathbf{z}(n))$, and assume the notation of Lemma 3.4.2. Then $\{\mathbf{z}(n)\} \subseteq L^0_{\mathbb{C}^k}(f(\mathbf{z}(0)))$. The vectors $\mathbf{p}(n)$ satisfy (3.4.16), so by Lemma 3.4.3 the sequence of iterates $\{\mathbf{z}(n)\}$ converges to $\hat{\mathbf{z}}$, and the rate of convergence is at least R-linear. Thus we have proved Theorem 3.4.4. □

To apply the one-step Newton steplength algorithm to the Newton's method or pseudo-Newton's method backpropagation algorithm for complex-valued holomorphic multilayer perceptrons, at the nth iteration in the training process, the one-step Newton steplength for the pth step in the backpropagation ($1 \leq p \leq L$) is

$$\mu_p(n) = \frac{-\mathrm{Re}\left(\frac{\partial E}{\partial \mathbf{w}}\Delta\mathbf{w}\right)}{\mathrm{Re}\left\{(\Delta\mathbf{w})^*\mathcal{H}_{\mathbf{ww}}\Delta\mathbf{w} + (\Delta\mathbf{w})^*\mathcal{H}_{\overline{\mathbf{w}}\mathbf{w}}\overline{\Delta\mathbf{w}}\right\}}, \tag{3.4.22}$$

where $\Delta\mathbf{w} = \Delta\mathbf{w}^{(p-1)}$ is the weight update for the pth layer of the network given by Theorem 3.2.1 or Corollary 3.3.1, respectively, and $\mathbf{w} = \mathbf{w}^{(p-1)}$. (Recall (2.3.2), so that here $\mathbf{p}(n) = -\Delta\mathbf{w}^{(p-1)}$ in (3.4.17).) For the pseudo-Newton's method backpropagation, we set $\mathcal{H}_{\overline{\mathbf{w}^{(p-1)}}\mathbf{w}^{(p-1)}} = \mathcal{H}_{\mathbf{w}^{(p-1)}\overline{\mathbf{w}^{(p-1)}}} = 0$ in (3.4.18) to obtain the pseudo-Newton updates $\Delta\mathbf{w}^{(p-1)}$ given in Corollary 3.3.1, but leave $\mathcal{H}_{\overline{\mathbf{w}^{(p-1)}}\mathbf{w}^{(p-1)}}$ as calculated in Theorem 3.2.1 in (3.4.22). In theory, for the nth iteration in the training process, we should choose the underrelaxation factor $\omega_p(n)$ for the pth step in the backpropagation ($1 \leq p \leq L$) according to (3.4.19) and (3.4.20). However, in practical application it suffices to take the underrelaxation factors to be constant and they may be chosen experimentally to yield convergence of the error function (see our results in Section VII). It is also not necessary in practical application to verify all the conditions of Theorem 3.4.4. In particular we may assume that the error function has a stationary point sufficiently close to the initial weights since the initial weights were chosen specifically to be "nearby" a stationary point, and that the stationary point is unique in the appropriate compact level set of the initial weights since the set of zeros of the error function has measure zero.

3.5 Experiments

To test the efficiency of the algorithms in the previous sections, we will compare the results of applying the gradient descent method, Newton's method, and the pseudo-Newton's method to a holomorphic MLP trained with data from the real-valued exclusive-or (XOR) problem (see Table 3.1). Note that the complex-valued XOR problem has different criteria for the data set [42]. We use the real-valued XOR problem as we desire a complex-valued network to process real as well as complex data.

The XOR problem is frequently encountered in the literature as a test case for backpropagation algorithms [39]. A multilayer network is required to solve it: without hidden units the network is unable to distinguish overlapping input patterns which map to different output patterns, e.g. $(0, 0)$ and $(1, 0)$ [45]. We use a two-layer network with $m = 2$ input nodes, $K = 4$ hidden nodes, and $C = 1$ output nodes. Any Boolean function of m variables can be trained to a two-layered real-valued neural network with 2^m hidden units. Modeling after the real case we choose $K = 2^m$, although this could perhaps be accomplished with fewer hidden units, as 2^{m-1} is a smaller upper bound for real-valued neural networks [22]. Some discussion of approximating Boolean functions, including the XOR and parity problems, using complex-valued neural networks is given in [37].

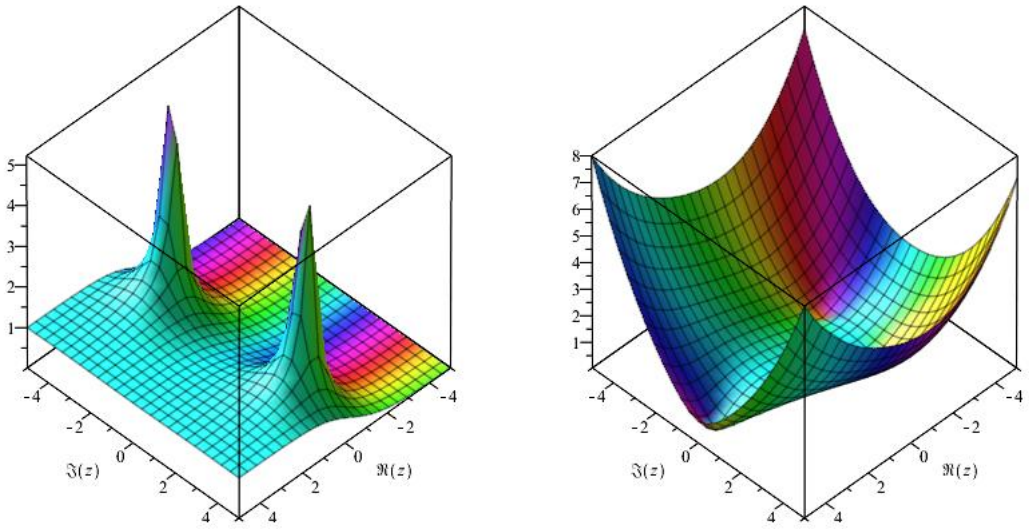
In our experiments, the activation functions are taken to be the same for both the hidden and output layers of the network. The activation function is either the sigmoidal function or its third degree Taylor polynomial approximation

$$g(z) = \frac{1}{1 + \exp(-z)} \text{ or } T(z) = \frac{1}{2} + \frac{1}{4}z - \frac{1}{48}z^3.$$

Note that one can take a higher degree Taylor polynomial approximation, but this is

Input Pattern	Output
0 0	0
1 0	1
0 1	1
1 1	0

Table 3.1: XOR Training Set



Sigmoidal Function Taylor Polynomial

Figure 3.1: The sigmoidal function (left) has two poles in a region near 0, while a Taylor polynomial approximation (right) of the sigmoidal function is bounded on the same region.

sufficient for our purposes. Notice that while $g(z)$ has poles near zero, the polynomial $T(z)$ is analytic on the entire complex plane and bounded on bounded regions (see Figure 3.1).

For each activation function we trained the network using the gradient descent backpropagation algorithm, the Newton backpropagation algorithm, and the pseudo-Newton backpropagation algorithm. The real and imaginary parts of the initial weights for each trial were chosen randomly from the interval $[-1, 1]$ according to a uniform distribution. In each case the network was trained to within 0.001 error. One hundred trials were performed for each activation function and each backpropagation algorithm (note that the same set of random initial weights was used for each set of trials). For the trials using the gradient descent backpropagation algorithm, a constant learning rate (μ) was used. It is known that for the gradient descent algorithm for real-valued neural networks, some learning rates will result in nonconvergence of the error function [31]. There is experimental evidence that for elementary transcendental activation functions used in complex-valued neural networks, sensitivity of the gradient descent algorithm to the choice of the learning rate can result in nonconvergence of the error function as well, and this is not necessarily affected by changes in the initial weight distribution [42]. To avoid these problems, a learning rate of $\mu = 1$ was chosen both to guarantee convergence and to

Activation Function	**Training Method**	**Learning Rate (μ)**	**Underrelaxation Factor (ω)**	**Number of Successful Trials**	**Average Number of Iterations***
Sigmoidal	Gradient Descent	$\mu = 1$	None	93	1258.9
Sigmoidal	Newton	One-Step Newton	$\omega = 0.5$	5	7.0
Sigmoidal	Pseudo-Newton	One-Step Newton	$\omega = 0.5$	78	7.0
Polynomial	Gradient Descent	$\mu = 1$	None	93	932.2
Polynomial	Newton	One-Step Newton	$\omega = 0.5$	53	107.9
Polynomial	Pseudo-Newton	One-Step Newton	$\omega = 0.5$	99	23.7

*Over the successful trials.

Table 3.2: XOR Experiment Results: Successful Trials

yield fast convergence (as compared to other values of μ). For the trials using the Newton and pseudo-Newton backpropagation algorithms, a variable learning rate (steplength) was chosen according to the one-step Newton steplength algorithm (Theorem 3.4.4) to control the problem of "overshooting" of the iterates and non-convergence of the error function when a fixed learning rate was used. For both the Newton and pseudo-Newton trials, a constant underrelaxation factor of $\omega = 0.5$ was used; this was chosen to yield the best chance for convergence of the error function. The results are summarized in Table 3.2.

Over the successful trials, the polynomial activation function performed just as well as the traditional sigmoidal function for the gradient descent backpropagation algorithm and yielded more successful trials than the sigmoidal function for the Newton and pseudo-Newton backpropagation algorithms. We define a successful trial to be one in which the error function dropped below 0.001. We logged four different types of unsuccessful trials (see Table 3.3). Convergence of the error function to a local minimum occurred when, after at least 50,000 iterations for gradient descent and 5,000 iterations for the Newton and pseudo-Newton algorithms, the error function remained above 0.001 but had stabilized to within 10^{-10} between successive iterations. This occurred more frequently in the Newton's method trails than the gradient descent trials, which was expected due to the known sensitivity of Newton's method to the initial points. A blow up of the error function occurred when, after the same minimum number of iterations as above, the error function had increased

Activation Function	**Training Method**	**Local Minimum**	**Blow Up**	**Undefined Floating Point**	**Singular Matrix**	**Total Unsuccessful Trials**
Sigmoidal	Gradient Descent	1	0	6	N/A	7
Sigmoidal	Newton	0	0	68	27	95
Sigmoidal	Pseudo-Newton	0	0	14	8	22
Polynomial	Gradient Descent	0	0	7	N/A	7
Polynomial	Newton	26	2	2	17	47
Polynomial	Pseudo-Newton	1	0	0	0	1

Table 3.3: XOR Experiment Results: Unsuccessful Trials

to above 10^{10}. The final value of the error function was sometimes an undefined floating point number, probably the result of division by zero. This occurred less frequently with the polynomial activation function than with the sigmoidal activation function. Finally, the last type of unsuccessful trial resulted from a singular Hessian matrix (occurring only in the Newton and pseudo-Newton trials). This, necessarily, halted the backpropagation process, and occurred less frequently with the polynomial activation function than with the sigmoidal activation function.

As for efficiency, the Newton and pseudo-Newton algorithms required significantly fewer iterations of the backpropagation algorithm to train the network than the gradient descent method for each activation function. In addition to producing fewer unsuccessful trials, the pseudo-Newton algorithm yielded a lower average number of iterations than the Newton algorithm for the polynomial activation function and the same average number of iterations as the Newton algorithm for the sigmoidal activation function. The network with polynomial activation function trained using the pseudo-Newton algorithm produced the fewest unsuccessful trials. Overall, we conclude that the use of the polynomial activation function yields more consistent convergence of the error function than the use of the sigmoidal activation function, and the use of the Newton and pseudo-Newton algorithms yields significantly fewer training iterations than the use of the gradient descent method.

Chapter 4

The Singular Hessian Matrix Problem

As we have seen with the XOR example, we encounter two significant problems when training a complex-valued neural network using the Newton and pseudo-Newton backpropagation algorithms. The existence of singular Hessian matrices results in the halting of the backpropagation algorithm when such a singular matrix is encountered, and convergence of the error function to a local minimum results in the network not being properly trained (see Table 3.3). In this chapter, we focus on the singular Hessian matrix problem as it arises in the minimization of real-valued complex functions using Newton's method. The existence of local minima will be addressed in Chapter 5.

Let $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ be $\mathbb{R}$-differentiable. For the remainder of this chapter, we desire to restrict our attention to functions f that can be viewed as polynomial maps. The Open Mapping Theorem states that the image of any non-constant analytic function defined on an open set in the complex plane is open [11]. Since the real line is closed in the complex plane and, for any open set $U \subseteq \Omega$ we have $f(U) \subseteq \mathbb{R}$, f cannot be a polynomial function as viewed with domain $\Omega \subseteq \mathbb{C}^k$. So, consider f as a function $f(\mathbf{x}, \mathbf{y}) : D \subseteq \mathbb{R}^k \to \mathbb{R}$, where

$$D := \left\{ \left(\begin{array}{c} \mathbf{x} \\ \mathbf{y} \end{array} \right) \middle| \begin{array}{c} \mathbf{x}, \mathbf{y} \in \mathbb{R}^k \\ \mathbf{z} = \mathbf{x} + i\mathbf{y} \in \Omega \end{array} \right\} \subseteq \mathbb{R}^{2k},$$

and suppose f is a polynomial map. Assume each variable $x_1, ..., x_k, y_1, ..., y_k$ occurs in f. Denote by

$$\mathbb{R}[\mathbf{x}, \mathbf{y}] = \mathbb{R}[x_1, ..., x_k, y_1, ..., y_k]$$

the polynomial ring in the $2k$ variables $x_1, ..., x_k, y_1, ..., y_k$, so that $f \in \mathbb{R}[\mathbf{x}, \mathbf{y}]$.

Consider the minimization of f via the minimization algorithm with sequence of iterates $\{\mathbf{z}(n)\}$ given recursively by

$$\mathbf{z}(n+1) = \mathbf{z}(n) - \omega(n)\mu(n)\mathbf{p}(n), \ n = 0, 1, ..., \tag{4.0.1}$$

where the $\mathbf{p}(n)$ are the nonzero complex Newton updates

$$\begin{aligned}\mathbf{p}(\mathbf{z}(n)) = &- \left[\mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n)) - \mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z}(n))\mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}(\mathbf{z}(n))^{-1}\mathcal{H}_{\mathbf{z}\overline{\mathbf{z}}}(\mathbf{z}(n))\right]^{-1} \\ &\cdot \left[\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z}(n))\mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}(\mathbf{z}(n))^{-1}\left(\frac{\partial f}{\partial \overline{\mathbf{z}}}(\mathbf{z}(n))\right)^{*} - \left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n))\right)^{*}\right],\end{aligned} \tag{4.0.2}$$

and the $\mu(n) \in \mathbb{C}$. Theorem 3.4.4 gives conditions on the steplengths $\mu(n)$ and the underrelaxation factors $\omega(n)$ that guarantee convergence of the iterates to a local minimum. However, at each iteration, this minimization algorithm requires inversion of the matrices $\mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}(\mathbf{z}(n)) = \overline{\mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n))}$ and the Schur complement

$$\widetilde{\mathcal{H}_{\mathbf{zz}}}(\mathbf{z}(n)) = \mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n)) - \mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z}(n))\mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}(\mathbf{z}(n))^{-1}\mathcal{H}_{\mathbf{z}\overline{\mathbf{z}}}(\mathbf{z}(n)). \tag{4.0.3}$$

If either of these matrices becomes singular, the iteration halts and the algorithm must be restarted with a new initial iterate $\mathbf{z}(0)$. In this chapter, we construct an adaptive underrelaxation factor algorithm give by Theorem 4.1.1 that avoid the singularity of the Hessian matrices by imposing restrictions on the choice of the underrelaxation factors used in the minimization of a real-valued complex polynomial function using Newton's method. Corollary 4.1.2 gives an adaptive underrelaxation factor algorithm for the minimization of a real-valued complex polynomial function using the pseudo-Newton method based on Theorem 4.1.1. We test the pseudo-Newton algorithm with adaptive underrelaxation factors on a neuron and a small-scale MLP trained with the XOR dataset, and we find evidence that our algorithms do in fact significantly decreases the number of singular matrix errors from the number of such errors seen when the Newton and pseudo-Newton algorithms

are applied with constant underrelaxation factors.

4.1 An Adaptive Underrelaxation Factor Algorithm for Minimization of Real-Valued Complex Polynomial Maps via Newton's Method

Consider the minimization of a polynomial map $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ via Newton's method with sequence of iterates given by (4.0.1) and (4.0.2). Suppose f has total degree $\text{totdeg}(f) = d$. We first transform the iterates (4.0.1) using the $\mathbb{CR}$-calculus [29]. Define $\mathbf{c} := (\mathbf{z}, \overline{\mathbf{z}})^T \in \mathbb{C}^{2k}$ for $\mathbf{z} \in \mathbb{C}^k$. Then

$$\frac{\partial f}{\partial \mathbf{c}} = \left(\frac{\partial f}{\partial \mathbf{z}}, \frac{\partial f}{\partial \overline{\mathbf{z}}} \right).$$

Define

$$\mathcal{H}_{\mathbf{cc}} = \frac{\partial}{\partial \mathbf{c}} \left(\frac{\partial f}{\partial \mathbf{c}} \right)^* = \begin{pmatrix} \mathcal{H}_{\mathbf{zz}} & \mathcal{H}_{\overline{z}\mathbf{z}} \\ \mathcal{H}_{\mathbf{z}\overline{\mathbf{z}}} & \mathcal{H}_{\overline{\mathbf{zz}}} \end{pmatrix}.$$

Let $\mathbf{c}(n) = (\mathbf{z}(n), \overline{\mathbf{z}(n)})^T$ for $n = 0, 1, ...$ Assuming the Hessian matrices $\mathcal{H}_{\overline{\mathbf{zz}}}$, $\widetilde{\mathcal{H}_{\mathbf{zz}}}$, and $\mathcal{H}_{\mathbf{cc}}$ are invertible at each stage of the iteration, we can rewrite (4.0.1) equivalently using the iterates $\mathbf{c}(n) \in \mathbb{C}^{2k}$ as:

$$\mathbf{c}(n+1) = \mathbf{c}(n) - \omega(n)\mu(n)\mathbf{q}(n), \ n = 0, 1, ..., \tag{4.1.1}$$

where the $\mathbf{q}(n)$ are the nonzero complex Newton updates

$$\mathbf{q}(n) = -\mathcal{H}_{\mathbf{cc}}(\mathbf{c}(n))^{-1} \left(\frac{\partial f}{\partial \mathbf{c}}(\mathbf{c}(n)) \right)^*. \tag{4.1.2}$$

Viewing the Newton updates in this manner, we wish to guarantee the nonsingularity of the single matrix $\mathcal{H}_{\mathbf{cc}}(\mathbf{c}(n))$ at each iteration.

Let $\mathbf{r} = (\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{2k}$ for $\mathbf{z} = \mathbf{x} + i\mathbf{y} \in \mathbb{C}^k$. If $\mathcal{H}_{\mathbf{rr}}$ denotes the real Hessian matrix

$$\mathcal{H}_{\mathbf{rr}} = \frac{\partial}{\partial \mathbf{r}} \left(\frac{\partial f}{\partial \mathbf{r}} \right)^T = \left(\frac{\partial}{\partial \mathbf{x}}, \frac{\partial}{\partial \mathbf{y}} \right) \left(\frac{\partial f}{\partial \mathbf{x}}, \frac{\partial f}{\partial \mathbf{y}} \right)^T = \begin{pmatrix} \mathcal{H}_{\mathbf{xx}} & \mathcal{H}_{\mathbf{xy}} \\ \mathcal{H}_{\mathbf{yx}} & \mathcal{H}_{\mathbf{yy}} \end{pmatrix},$$

then we have the relation

$$\mathcal{H}_{\mathbf{rr}} = J^* \mathcal{H}_{\mathbf{cc}} J, \tag{4.1.3}$$

where J is the complex $2k \times 2k$ block matrix

$$J := \begin{pmatrix} I & iI \\ I & -iI \end{pmatrix} \in M_{2k \times 2k}(\mathbb{C}).$$

Since $f : D \subseteq \mathbb{R}^{2k} \to \mathbb{R}$ is a polynomial map of total degree d, each second-order partial derivative

$$\frac{\partial}{\partial x_h} \left(\frac{\partial f}{\partial x_i} \right), \ \frac{\partial}{\partial y_l} \left(\frac{\partial f}{\partial x_i} \right), \ \frac{\partial}{\partial x_h} \left(\frac{\partial f}{\partial y_j} \right), \text{ and } \frac{\partial}{\partial y_l} \left(\frac{\partial f}{\partial y_j} \right),$$

where $i, j, h, l = 1, ..., k$, is a polynomial of total degree 0 or $d - 2$, depending on the degree of f in each respective variable. As these partial derivatives form the entries of the Hessian matrix $\mathcal{H}_{\mathbf{rr}}$, we have that $\mathcal{H}_{\mathbf{rr}}$ is a matrix with entries in the polynomial ring $\mathbb{R}[\mathbf{x}, \mathbf{y}]$. Using (4.1.3), we see that $\mathcal{H}_{\mathbf{cc}}$ is a matrix with entries in the polynomial ring $\mathbb{C}[\mathbf{x}, \mathbf{y}]$, with each entry having total degree 0 or $d - 2$. Then the determinant $\det \mathcal{H}_{\mathbf{cc}}$ is a polynomial in $\mathbb{C}[\mathbf{x}, \mathbf{y}]$.

Let $\tilde{d} = \text{totdeg}(\det \mathcal{H}_{\mathbf{cc}})$. Since each variable $x_1, ..., x_k, \ y_1, ..., y_k$ occurs in f, we cannot have an entire row or column in $\mathcal{H}_{\mathbf{rr}}$ (and hence in $\mathcal{H}_{\mathbf{cc}}$) be zero. Thus $\det \mathcal{H}_{\mathbf{cc}} \not\equiv 0$, and each product that occurs in the complete expansion of the determinant by permutations has total degree 0 or $(d-2)^{2k}$, so allowing for cancellation of terms, we have that

$$\tilde{d} = \text{totdeg}(\det \mathcal{H}_{\mathbf{cc}}) \leq (d-2)^{2k}.$$

Notice that the Hessian matrix $\mathcal{H}_{\mathbf{cc}}$ with entries in $\mathbb{C}[\mathbf{x}, \mathbf{y}]$ is singular at the nth

iterate if and only if the polynomial $\det \mathcal{H}_{\mathbf{cc}} = 0$ at that iterate. Since

$$\mathbf{c} = (\mathbf{z}, \overline{\mathbf{z}})^T = (\mathbf{x} + i\mathbf{y}, \mathbf{x} - i\mathbf{y})^T,$$

we can relax our notation by denoting $\mathcal{H}_{\mathbf{cc}}(\mathbf{c}(n)) = \mathcal{H}_{\mathbf{cc}}(\mathbf{z}(n)) = \mathcal{H}_{\mathbf{cc}}(\mathbf{x}(n), \mathbf{y}(n))$ and noting that $\mathcal{H}_{\mathbf{cc}}$ is singular if and only if

$$\det \mathcal{H}_{\mathbf{cc}}(\mathbf{x}(n), \mathbf{y}(n)) = 0.$$

Thus, to guarantee that the Hessian matrix $\mathcal{H}_{\mathbf{cc}}$ is nonsingular at the nth iteration, we give conditions to ensure that $(\mathbf{x}(n), \mathbf{y}(n))$ does not lie in the variety $V(\det \mathcal{H}_{\mathbf{cc}}) \subseteq \mathbb{R}^{2k}$.

Before we give the proposed algorithm, we state the following result, as given in [41]. Let

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = 0 \tag{4.1.4}$$

be a polynomial equation of degree n, where the a_j, $j = 1, ..., n$ are real or complex coefficients with $a_n \neq 0$. If x is a root of (4.1.4), then

$$|x| \leq 1 + \frac{A}{|a_n|}, \tag{4.1.5}$$

where $A = \max\{|a_0|, |a_1|, ..., |a_{n-1}|\}$. If, in addition, $a_0 \neq 0$, then $x = 0$ is not a root of (4.1.4). So if x is a root of (4.1.4) with $a_0 \neq 0$, then we can write $x = 1/y$ with $y \neq 0$, so:

$$0 = \sum_{i=0}^{n} a_i x^i = \sum_{i=0}^{n} a_i \left(\frac{1}{y}\right)^i = \sum_{i=0}^{n} \frac{a_i}{y^i}.$$

Multiplying both sides by y^n yields

$$0 = y^n \sum_{i=0}^{n} \frac{a_i}{y^i} = \sum_{i=0}^{n} a_i y^{n-i} = \sum_{j=0}^{n} a_{n-j} y^j,$$

with $a_{n-n} = a_0 \neq 0$. Using (4.1.5), we have that

$$|y| \leq 1 + \frac{A'}{|a_0|},$$

where $A' = \max\{|a_n|, |a_{n-1}|, ..., |a_1|\}$. Then

$$|x| = \frac{1}{|y|} \geq 1 + \frac{A'}{|a_0|} = \frac{|a_0| + A'}{|a_0|}. \tag{4.1.6}$$

Theorem 4.1.1 (Underrelaxation Factors for the Complex Newton Algorithm for Minimization of Polynomial Functions)**.** *Let $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ be $\mathbb{R}$-differentiable such that $f : D \subseteq \mathbb{R}^{2k} \to \mathbb{R}$ is a polynomial map. Suppose the determinant $\det \mathcal{H}_{\mathbf{cc}} : D \subseteq \mathbb{C}^{2k} \to \mathbb{C}$ has total degree $\tilde{d}$. Let $\mathbf{z}(0) \in \Omega$ such that $\det \mathcal{H}_{\mathbf{cc}}(\mathbf{z}(0)) \neq 0$. Consider the iteration*

$$\mathbf{z}(n+1) = \mathbf{z}(n) + \omega(n)\mu(n)\Delta\mathbf{z}(n), \ n = 0, 1, ..., \tag{4.1.7}$$

where the $\Delta\mathbf{z}(n)$ are the nonzero complex Newton updates

$$\Delta\mathbf{z}(n) = -\widetilde{\mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n))}^{-1} \left[\mathcal{H}_{\overline{\mathbf{z}}\mathbf{z}}(\mathbf{z}(n)) \mathcal{H}_{\overline{\mathbf{z}}\overline{\mathbf{z}}}(\mathbf{z}(n))^{-1} \left(\frac{\partial f}{\partial \overline{\mathbf{z}}}(\mathbf{z}(n)) \right)^* - \left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n)) \right)^* \right], \tag{4.1.8}$$

and the steplengths $\mu(n) \in \mathbb{C}$. Let

$$\Phi_n(\xi) := \det \mathcal{H}_{\mathbf{cc}}(\mathbf{z}(n) + \xi\mu(n)\Delta\mathbf{z}(n)) \in \mathbb{C}[\xi], \ n = 1, 2, 3, ... \tag{4.1.9}$$

Suppose

$$\Phi_n(\xi) = \sum_{l=0}^{\tilde{d}} b_l^{(n)} \xi^l, \ n = 1, 2, 3, ..., \tag{4.1.10}$$

and let

$$\hat{\xi_n} = \frac{|b_0^{(n)}|}{|b_0^{(n)}| + B^{(n)}}, \ \text{where } B^{(n)} = \max_{1 \leq l \leq \tilde{d}} |b_l^{(n)}|. \tag{4.1.11}$$

If the underrelaxation factors $\omega(n)$ satisfy

$$0 \leq \omega(n) \leq \hat{\xi_n}, \tag{4.1.12}$$

then the Hessian matrices $\mathcal{H}_{\mathbf{cc}}(\mathbf{z}(n+1))$, $n = 1, 2, ...,$ are nonsingular.

Proof. Let $\mathbf{P} = (p_1, ..., p_k) \in \mathbb{C}^k$ be a point and $\mathbf{v} = \langle v_1, ..., v_k \rangle \in \mathbb{C}^k$ a vector. Suppose $p_j = p_{j1} + ip_{j2}$ and $v_j = v_{j1} + iv_{j2}$, with $p_{j1}, p_{j2}, v_{j1}, v_{j2} \in \mathbb{R}$, for $j = 1, ..., k$. If $\mathbf{P}_1 = (p_{11}, ..., p_{k1})$, $\mathbf{P}_2 = (p_{12}, ..., p_{k2}) \in \mathbb{R}^{2k}$ and $\mathbf{v}_1 = \langle v_{11}, ..., v_{k1} \rangle$, $\mathbf{v}_2 = \langle v_{12}, ..., v_{k2} \rangle \in \mathbb{R}^{2k}$, then we can view $\mathbf{P}$ and $\mathbf{v}$ as elements of $\mathbb{R}^{2k}$ as $\mathbf{P} = (\mathbf{P}_1, \mathbf{P}_2) \in \mathbb{R}^{2k}$ and $\mathbf{v} = \langle \mathbf{v}_1, \mathbf{v}_2 \rangle \in \mathbb{R}^{2k}$. The line in $\mathbb{R}^{2k}$ through $\mathbf{P}$ in the direction of $\mathbf{v}$ is given by [16]:

$$L := \{(\mathbf{P}_1 + \xi \mathbf{v}_1, \mathbf{P}_2 + \xi \mathbf{v}_2) \,|\, \xi \in \mathbb{R}\} \subseteq \mathbb{R}^{2k}. \tag{4.1.13}$$

We proceed with the proof by induction on n. Set $\mathbf{P} = \mathbf{z}(0)$. Since $\det \mathcal{H}_{\mathbf{cc}}(\mathbf{z}(0)) \neq 0$, the Hessian $\mathcal{H}_{\mathbf{cc}}(\mathbf{z}(0))$ is nonsingular. Thus the complex Newton update $\Delta\mathbf{z}(0)$ is well-defined. Let $\mathbf{v} = \mu(0)\Delta\mathbf{z}(0)$, and for ease of notation view $\mathbf{P}$ and $\mathbf{v}$ as points in $\mathbb{R}^{2k}$, as above. Denote by $L(0)$ the line in $\mathbb{R}^{2k}$ through $\mathbf{z}(0)$ in the direction of $\mathbf{v}$ given by (4.1.13):

$$L(0) := \{\mathbf{z}(0) + \xi\mu(0)\Delta\mathbf{z}(0) \,|\, \xi \in \mathbb{R}\} \subseteq \mathbb{R}^{2k}.$$

Notice that for $\omega(0) \in \mathbb{R}$, the iterate

$$\mathbf{z}(1) = \mathbf{z}(0) + \omega(0)\mu(0)\Delta\mathbf{z}(0) \in L(0).$$

To guarantee that $\mathcal{H}_{\mathbf{cc}}(\mathbf{z}(1))$ is nonsingular, it is sufficient to show that $\det \mathcal{H}_{\mathbf{cc}}(\mathbf{z}(1)) \neq 0$, that is, $\mathbf{z}(1) \notin V(\det \mathcal{H}_{\mathbf{cc}})$.

To this end, points in $L(0) \cap V(\det \mathcal{H}_{\mathbf{cc}})$ satisfy

$$det\mathcal{H}_{\mathbf{cc}}(\mathbf{z}(0) + \xi\mu(0)\Delta\mathbf{z}(0)) = 0.$$

Define $\Phi_1 \in \mathbb{R}[\xi]$ as in (4.1.9). Then, since the total degree of $\det \mathcal{H}_{\mathbf{cc}} \in \mathbb{C}[\mathbf{x}, \mathbf{y}]$ is $\tilde{d}$, Φ_1 has degree $\tilde{d}$, and we can expand Φ_1 as in (4.1.10):

$$\Phi_1(\xi) = \sum_{l=0}^{\tilde{d}} b_l^{(1)} \xi^l. \tag{4.1.14}$$

Since $\mathbf{z}(0) \notin V(\det \mathcal{H}_{\mathbf{cc}})$, $\xi = 0$ is not a root of Φ_1, so the constant term $b_0^{(1)} \neq 0$.

Using (4.1.6), if ξ is a root of (4.1.14), then

$$|\xi| \geq \frac{|b_0^{(1)}|}{|b_0^{(1)}| + B^{(1)}}, \text{ where } B^{(1)} = \max_{1 \leq l \leq \tilde{d}} |b_l^{(1)}|.$$

If we set

$$\hat{\xi_1} = \frac{|b_0^{(1)}|}{|b_0^{(1)}| + B^{(1)}},$$

and if we choose $\omega(0) \in \mathbb{R}$ with $0 < \omega(0) < \hat{\xi_1}$, then

$$\mathbf{z}(0) + \omega(0)\mu(0)\Delta\mathbf{z}(0) \in L(0) - V(\det \mathcal{H}_{\mathbf{cc}}).$$

Thus $\det \mathcal{H}_{\mathbf{cc}}(\mathbf{z}(1)) \neq 0$, so the Hessian matrix $\mathcal{H}_{\mathbf{cc}}$ is nonsingular.

Assuming $\det \mathcal{H}_{\mathbf{cc}}(\mathbf{z}(n)) \neq 0$ and repeating this argument for the nth iteration of Newton's method, it follows that given the choice (4.1.12) of underrelaxation factor $\omega(n)$ for the nth iteration, the Hessian matrix $\mathcal{H}_{\mathbf{cc}}(\mathbf{z}(n+1))$ is nonsingular. □

For the pseudo-Newton algorithm, we take $\mathcal{H}_{\bar{\mathbf{z}}\bar{\mathbf{z}}} = \mathcal{H}_{\mathbf{z}\bar{\mathbf{z}}} = 0$ and we obtain the following corollary.

Corollary 4.1.2 (Underrelaxation Factors for the Complex Pseudo-Newton Algorithm for Minimization of Polynomial Functions)**.** *Let $f : \Omega \subseteq \mathbb{C}^k \to \mathbb{R}$ be as in Theorem 4.1.1. Suppose the determinant $\det \mathcal{H}_{\mathbf{zz}} : D \subseteq \mathbb{C}^{2k} \to \mathbb{C}$ has total degree $\tilde{d}$. Let $\mathbf{z}(0) \in \Omega$ such that $\det \mathcal{H}_{\mathbf{zz}}(\mathbf{z}(0)) \neq 0$. Consider the iteration (4.1.7), where the $\Delta\mathbf{z}(n)$ are the nonzero complex pseudo-Newton updates*

$$\Delta\mathbf{z}(n) = -\mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n))^{-1} \left(\frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}(n)) \right)^*$$

and the steplengths $\mu(n) \in \mathbb{C}$. Define Φ_n by (4.1.9) and expand Φ_n as in (4.1.10). Define $\hat{\xi_n}$ as in (4.1.11). If the underrelaxation factors $\omega(n)$ satisfy

$$0 \leq \omega(n) \leq \hat{\xi_n},$$

then the Hessian matrices $\mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n+1))$, $n = 1, 2, ...,$ are nonsingular.

Proof. For the pseudo-Newton method,

$$\mathcal{H}_{\mathbf{cc}}(\mathbf{z}(n)) = \begin{pmatrix} \mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n)) & 0 \\ 0 & \mathcal{H}_{\overline{\mathbf{zz}}}(\mathbf{z}(n)) \end{pmatrix},$$

so

$$\begin{aligned} \det \mathcal{H}_{\mathbf{cc}}(\mathbf{z}(n)) &= \det \mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n)) \det \mathcal{H}_{\overline{\mathbf{zz}}}(\mathbf{z}(n)) \\ &= \det \mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n)) \overline{\det \mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n))} \\ &= \left|\det \mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n))\right|^2. \end{aligned}$$

Thus $\mathcal{H}_{\mathbf{cc}}(\mathbf{z}(n))$ is nonsingular if and only if $\mathcal{H}_{\mathbf{zz}}(\mathbf{z}(n))$ is nonsingular, so we may apply Theorem 4.1.1. □

4.2 Application: The Artificial Neuron

The simplest case of the artificial neural network is the artificial neuron, and we can gain some valuable perspective in applying Theorem 4.1.1 to this basic building block of the artificial neural network. Consider a complex-valued artificial neuron with m inputs (see Figure 2.1). As in Section 2.1, denote the inputs by $x_1, ..., x_m$ and the weights by $w_1, ..., w_m$. Let $g : \mathbb{C} \to \mathbb{C}$ be the activation function for the neuron, and suppose g is a polynomial of degree n with real coefficients given by

$$g(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0,$$

where $a_i \in \mathbb{C}$ for $i = 1, ..., n$ and $a_n \neq 0$. Then the output of the neuron is given by y, where

$$y = g\left(\sum_{i=1}^{m} w_i x_i\right).$$

We train the neuron with the training set $\{(z_{t1}, ..., z_{tm}, d_t) \,|\, t = 1, ..., N\}$ by applying the Newton or pseudo-Newton backpropagation algorithm (Theorem 3.2.1, Corollary

3.3.1, respectively) to the error function

$$E = \frac{1}{N}\sum_{t=1}^{N} |y_t - d_t|^2 = \frac{1}{N}\sum_{t=1}^{N} (y_t - d_t)\left(\overline{y_t} - \overline{d_t}\right).$$

Note that we can readily apply Theorem 3.2.1 or Corollary 3.3.1, since g has real coefficients, hence the condition $\overline{g(z)} = g(\overline{z})$ holds.

Let $\mathbf{w} = (w_1, ..., w_m)$ be the weight vector for the neuron. Using the Newton algorithm in the neuron case, the weight update is given by

$$\begin{aligned}\Delta\mathbf{w}(n) = -\omega(n)\mu(n)\left[\mathcal{H}_{\mathbf{ww}}(\mathbf{w}(n)) - \mathcal{H}_{\overline{\mathbf{w}}\mathbf{w}}(\mathbf{w}(n))\mathcal{H}_{\overline{\mathbf{ww}}}(\mathbf{w}(n))\mathcal{H}_{\mathbf{w}\overline{\mathbf{w}}}(\mathbf{w}(n))\right]^{-1} \\ \cdot \left[\mathcal{H}_{\overline{\mathbf{w}}\mathbf{w}}(\mathbf{w}(n))\mathcal{H}_{\overline{\mathbf{ww}}}(\mathbf{w}(n))^{-1}\left(\frac{\partial f}{\partial \overline{\mathbf{w}}}(\mathbf{w}(n))\right)^{*} - \left(\frac{\partial f}{\partial \mathbf{w}}(\mathbf{w}(n))\right)^{*}\right],\end{aligned}$$

where the $\mu(n)$ are the learning rates given by the complex one-step Newton step-length algorithm (Theorem 3.4.4), and the $\omega(n)$ are the underrelaxation factors. Let $w_j = w_{j1} + iw_{j2}$, $x_{tj} = x_{tj1} + ix_{tj2}$, and $d_t = d_{t1} + id_{t2}$ for $j = 1, ..., m$ and $t = 1, ..., N$. Since the activation function g is a polynomial in one variable with real coefficients, we have the following:

$$y_t = g\left(\sum_{j=1}^{m} w_j w_{tj}\right) = g\left(\sum_{j=1}^{m} (w_{j1} + iw_{j2})(x_{tj1} + ix_{tj2})\right)$$

and

$$\overline{y_t} = \overline{g\left(\sum_{j=1}^{m} (w_{j1} + iw_{j2})(x_{tj1} + ix_{tj2})\right)} = g\left(\sum_{j=1}^{m} (w_{j1} - iw_{j2})(x_{tj1} - ix_{tj2})\right),$$

so that

$$\begin{aligned} E &= \frac{1}{N}\sum_{t=1}^{N}(y_t - d_t)\left(\overline{y_t} - \overline{d_t}\right) \\ &= \frac{1}{N}\sum_{t=1}^{N}\left[g\left(\sum_{j=1}^{m}(w_{j1}+iw_{j2})(x_{tj1}+ix_{tj2})\right) - (d_{t1}+id_{t2})\right] \\ &\qquad \cdot \left[g\left(\sum_{j=1}^{m}(w_{j1}-iw_{j2})(x_{tj1}-ix_{tj2})\right) - (d_{t1}-id_{t2})\right] \end{aligned}$$

is a real-valued polynomial in $w_{11}, ..., w_{m1}, w_{12,}, ..., w_{m2}$ when viewed as a function of the real and imaginary parts of the weights. In particular, E has real coefficients (that is, $E \in \mathbb{R}[w_{11}, ..., w_{m1}, w_{12,}, ..., w_{m2}]$) when viewed as a polynomial in the real and imaginary parts of the weights, since

$$\overline{E} = \overline{\frac{1}{N}\sum_{t=1}^{N}(y_t - d_t)\left(\overline{y_t} - \overline{d_t}\right)} = \frac{1}{N}\sum_{t=1}^{N}\left(\overline{y_t} - \overline{d_t}\right)(y_t - d_t) = E.$$

Hence we can apply Theorem 4.1.1 and choose underrelaxation factors $\omega(n)$ satisfying (4.1.12).

We look at the particular case of training a neuron with the pseudo-Newton algorithm with underrelaxation factors given by Corollary 4.1.2, which allows for an easier and more direct application in simulations. Suppose we have a neuron with $m = 2$ inputs and activation function given by the third-degree Taylor polynomial

$$g(z) = \frac{1}{2} + \frac{1}{4}z - \frac{1}{48}z^3$$

to the standard sigmoidal activation function. We train the neuron using the training set given in Table 4.1. In order to see the improvement obtained in applying Corollary 4.1.2 to the pseudo-Newton backpropagation training algorithm, we com-

Input Vector	Output
$(1, 2)$	1
$(0, -1)$	0

Table 4.1: Artificial Neuron Training Set

pare the graphs of the singular matrix errors obtained in using constant versus adaptive underrelaxation factors when choosing initial weights for the backpropagation algorithm from cross sections of the complex weight space $\mathbb{C}^2$. We trained the neuron using the pseudo-Newton backpropagation algorithm with adaptive complex one-step Newton steplengths (Theorem 3.4.4).

Figures 4.1 and 4.2 show rectangular regions in $\mathbb{C}^2$ taken from the planes $\mathbb{R} \times \mathbb{R}\, i$ and $\mathbb{R} \times \mathbb{R}\, i$, respectively, which are color coded to represent initial weights corresponding to singular matrix errors and local minima. The horizontal axis represents the choice of w_1 from, respectively, the real line $\mathbb{R}$ and the imaginary line $\mathbb{R}\, i$, and the vertical axis represents the choice of w_2 from the imaginary line $\mathbb{R}\, i$. The magenta regions correspond to initial weights that result in singular matrix errors. The cyan regions correspond to initial weights that result in convergence of the error function to a local minimum. The white regions correspond to initial weights that result in successful training of the neuron, that is, convergence of the error function to a global minimum of zero. The graphs on the left represent the training of the neuron using a constant underrelaxation factor of $\omega = 0.5$, and the graphs on the right represent the training of the neuron using the adaptive underrelaxation factor algorithm given by Corollary 4.1.2.

We observe a significant decrease in the regions corresponding to singular matrix errors when applying our adaptive underrelaxation factor algorithm. This indicates experimental evidence that Corollary 4.1.2 does in fact decrease the frequency of singular matrix errors in training a complex neuron. It is interesting to note, however, an expansion of the regions corresponding to convergence of the error function to local instead of global minima. Avoidance of local minima is addressed in the next chapter. The fact that the pattern of the region of convergence to local minima obtained when applying this adaptive underrelaxation factor algorithm somewhat mimics the pattern of the region of singular matrix errors obtained when applying a constant underrelaxation factor warrants further investigation.

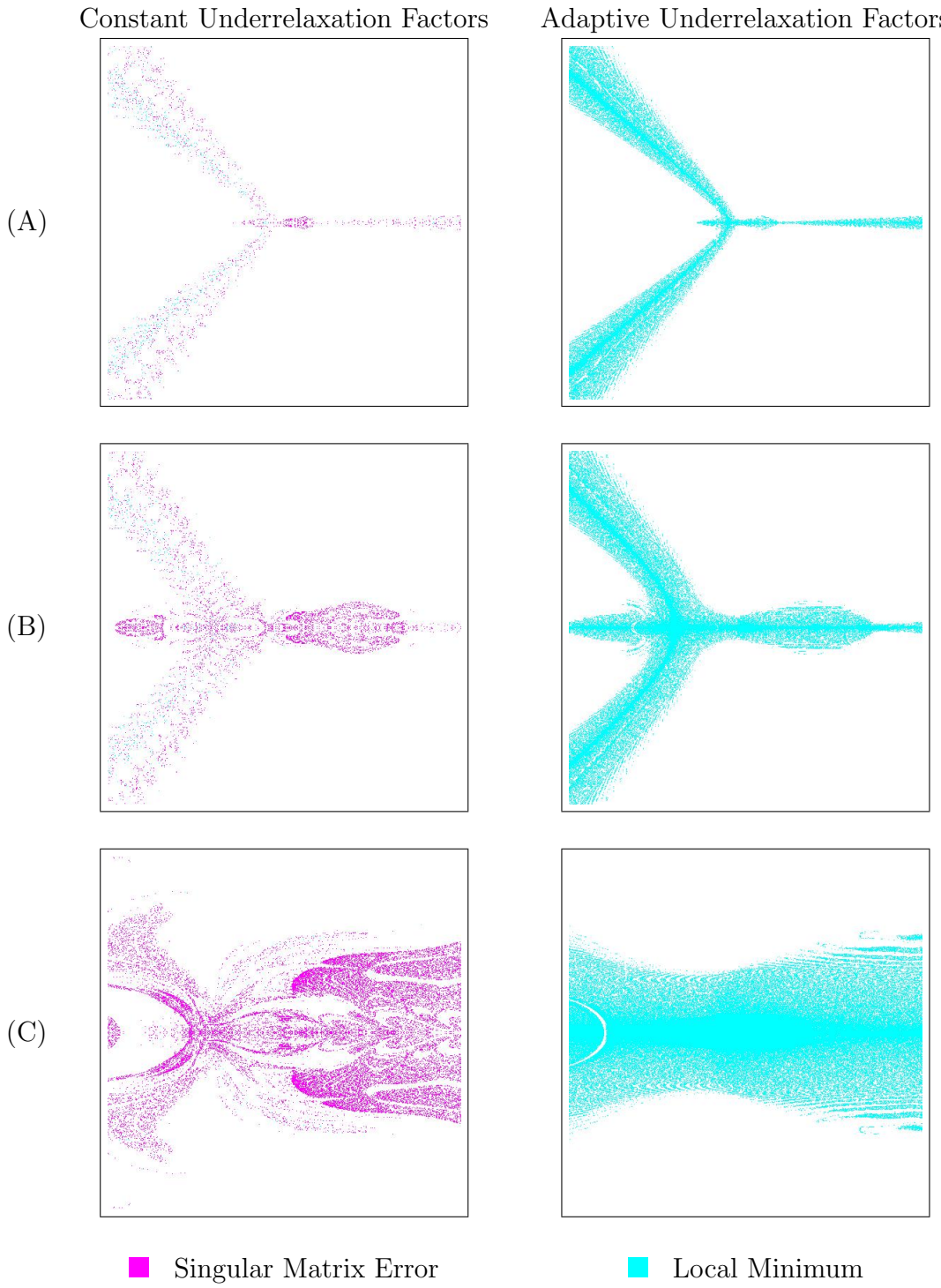


Figure 4.1: These graphs show the singular matrix errors and local minima obtained when choosing initial weights (w_1, w_2) from the regions (A) $[-20, 20] \times [-20, 20]\, i$, (B) $[-5, 5] \times [-5, 5]\, i$, and (C) $[-1, 1] \times [-1, 1]\, i$ for the complex neuron trained using Table 4.1. The horizontal axis represents the choice of w_1 from the real line $\mathbb{R}$, and the vertical axis represents the choice of w_2 from the imaginary line $\mathbb{R}\, i$. The singular matrix errors are eliminated in using the adaptive underrelaxation factors.

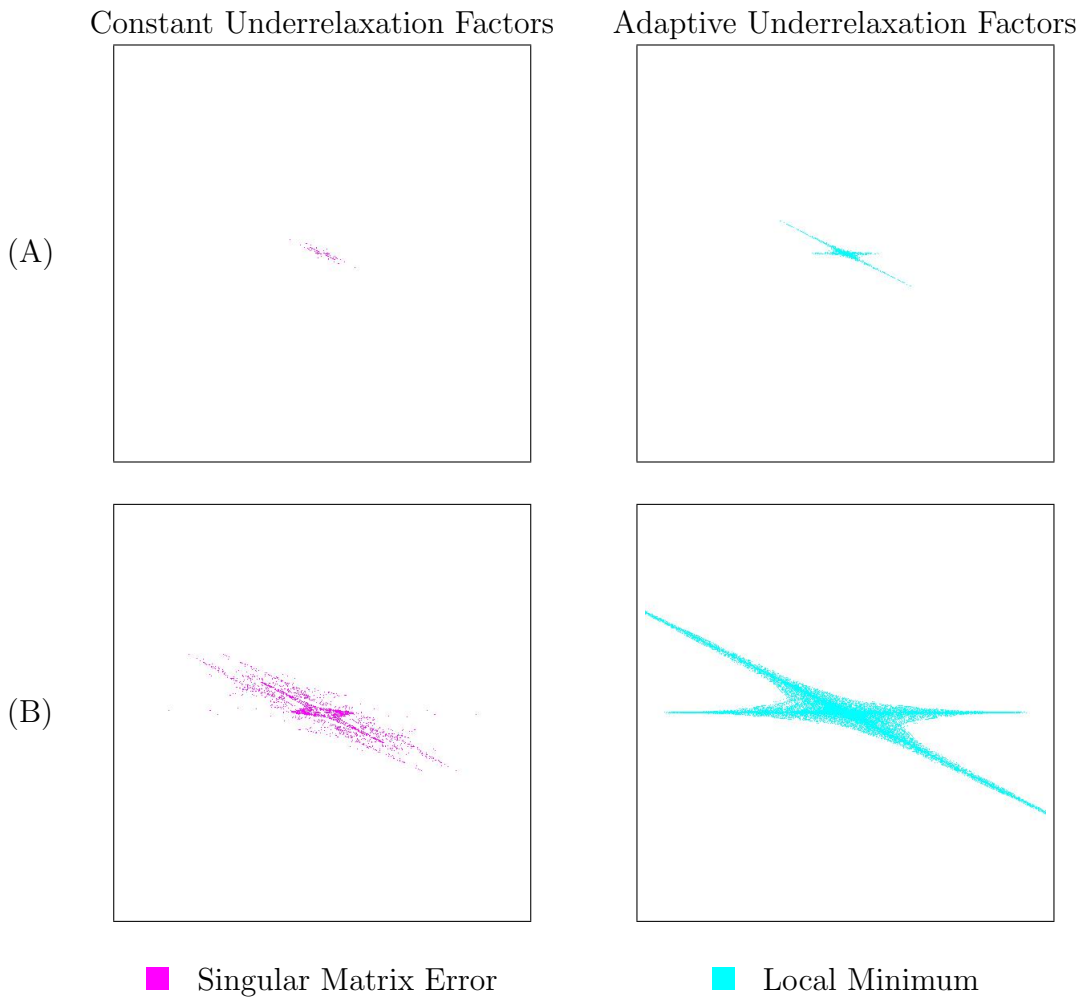


Figure 4.2: These graphs show the singular matrix errors and local minima obtained when choosing initial weights (w_1, w_2) from the regions (A) $[-20, 20]\,i \times [-20, 20]\,i$ and (B) $[-5, 5]\,i \times [-5, 5]\,i$ for the complex neuron trained using Table 4.1. The horizontal axis represents the choice of w_1 from the imaginary line $\mathbb{R}\,i$, and the vertical axis represents the choice of w_2 from the imaginary line $\mathbb{R}\,i$. The singular matrix errors are eliminated in using the adaptive underrelaxation factors.

4.3 Singular Hessian Matrices and the Multilayer Perceptron

We apply our underrelaxation factor algorithm to a complex-valued multilayer perceptron, and we test our algorithm on a two-layer MLP trained with the XOR data set (Table 3.1).

Definition 4.3.1. A **complex-valued polynomial MLP** is a complex-valued MLP in which the activation function in each layer of the network is a polynomial.

We employ the same notation as used in Chapter 2 for an L-layer holomorphic MLP (see Figure 2.2). For a polynomial MLP, we assume that the activation function of the pth layer of the network is a polynomial $g_p \in \mathbb{C}[z]$ for $p = 1, ..., L$. Let n_p be the degree of the polynomial g_p. In our setting, we assume that

$$g_p(z) = a_{n_p}^{(p)} z^{n_p} + a_{n_p-1}^{(p)} z^{n_p-1} + \cdots + a_1^{(p)} z + a_0^{(p)},$$

where $a_i^{(p)} \in \mathbb{R}$ for $i = 0, ..., n_p$, so that $\overline{g_p(z)} = g_p(\overline{z})$, allowing us to apply the Newton and pseudo-Newton backpropagation algorithms to train the network. The input layer of the network has m nodes and the output layer has C nodes, and we train the network using a training set with N data points

$$\{(z_{t1}, ..., z_{tm}, d_{t1}, ..., d_{tC} \,|\, t = 1, ..., N\},$$

where $(z_{t1}, ..., z_{tm})$ is the input vector for the tth training point which corresponds to the desired output vector $(d_{t1}, ..., d_{tC})$. We minimize the error function

$$E = \frac{1}{N} \sum_{t=1}^{N} \sum_{l=1}^{C} |y_{tl} - d_{tl}|^2 = \frac{1}{N} \sum_{t=1}^{N} \sum_{l=1}^{C} (y_{tl} - d_{tl}) \left(\overline{y_{tl}} - \overline{d_{tl}}\right),$$

where y_{tl} denotes the actual output of the network at node l when we apply the input vector corresponding to the tth training point for $l = 1, ..., C$ and $t = 1, ..., N$.

For $p = 1, ..., L$, set $w_{kl}^{(p-1)} = w_{kl1}^{(p-1)} + i w_{kl2}^{(p-1)}$, where $w_{kl1}^{(p-1)}, w_{kl2}^{(p-1)} \in \mathbb{R}$, for $k = 1, ..., K_p$ and $l = 1, ..., K_{p-1}$. Denote the weight vector for the pth layer of the network by $\mathbf{w}^{(p-1)}$ as in (2.3.1), and let $\mathbf{w}^{(p-1)} = \mathbf{w}_1^{(p-1)} + i\mathbf{w}_2^{(p-1)}$. Since the

activation function in each layer of the network is a polynomial, we have that for the pth layer, $p = 1, ..., L$, the output of the kth node, $k = 1, ..., K_p$, is

$$x_k^{(p)} = g_p\left(\sum_{l=1}^{K_{p-1}} \left(w_{kl1}^{(p-1)} + iw_{kl2}^{(p-1)}\right)\left(x_{l1}^{(p-1)} + ix_{l2}^{(p-1)}\right)\right),$$

where $x_l^{(p-1)} = x_{l1}^{(p-1)} + ix_{l2}^{(p-1)}$ for $l = 1, ..., K_{p-1}$ with $x_{l1}^{(p-1)}, x_{l2}^{(p-1)} \in \mathbb{R}$. Hence each $x_k^{(p)}$ for $k = 1, ..., K_p$ represents a polynomial in the variables $w_{kl1}^{(p-1)}, w_{kl2}^{(p-1)}$ for $l = 1, ..., K_{p-1}$. Recursively, then, a calculation similar to (4.2) shows that the error function is in fact a real-valued polynomial in the real variables

$$(\mathbf{w}_1^{(0)}, ..., \mathbf{w}_1^{(L-1)}, \mathbf{w}_2^{(0)}, ..., \mathbf{w}_2^{(L-1)})$$

corresponding to the real and imaginary parts of the weights. In particular, E has real coefficients, again because, similar to (4.2),

$$\overline{E} = \overline{\frac{1}{N}\sum_{t=1}^{N}\sum_{l=1}^{C}(y_{tl} - d_{tl})\left(\overline{y_{tl}} - \overline{d_{tl}}\right)} = \frac{1}{N}\sum_{t=1}^{N}\sum_{l=1}^{C}\left(\overline{y_{tl}} - \overline{d_{tl}}\right)(y_{tl} - d_{tl}) = E.$$

Hence we can apply Theorem 4.1.1 at each stage of the Newton or pseudo-Newton backpropagation algorithm.

We revisit the XOR example to compare our methodology to the current standard of using a constant underrelaxation factor. Tables 4.2 and 4.3 show our results in training the two-layer polynomial network given in Section 5 of Chapter 3 with activation function

$$g(z) = \frac{1}{2} + \frac{1}{4}z - \frac{1}{48}z^3$$

using the gradient descent method with constant learning rate ($\mu = 1$), Newton's method with one-step Newton steplengths and constant underrelaxation factors ($\omega = 0.5$), and the pseudo-Newton method with one-step Newton steplengths and both constant ($\omega = 0.5$) and adaptive underrelaxation factors. The real and imaginary parts of the initial weights were chosen randomly from the interval $[-1, 1]$ according to a uniform distribution, and the network was trained to within 0.05 error. Five hundred trials of each method were performed.

We see a marked improvement in using the pseudo-Newton method with adap-

Training Method	**Learning Rate (μ)**	**Underrelaxation Factor (ω)**	**Number of Successful Trials**	**Average Number of Iterations***
Gradient Descent	$\mu = 1$	N/A	499	419.8
Newton	One-Step Newton	$\omega = 0.5$	348	164.9
Pseudo-Newton	One-Step Newton	$\omega = 0.5$	500	11.1
Pseudo-Newton	One-Step Newton	Adaptive	420	91.9

*Over the successful trials.

Table 4.2: Adaptive Versus Constant Underrelaxation Factors for XOR Example: Successful Trials

Training Method	**Local Minimum**	**Blow Up**	**Undefined Floating Point**	**Singular Matrix**	**Total Unsuccessful Trials**
Gradient Descent ω N/A	0	0	1	0	1
Newton $\omega = 0.5$	99	6	2	45	152
Pseudo-Newton $\omega = 0.5$	0	0	0	0	0
Pseudo-Newton ω Adaptive	80	0	0	0	0

Table 4.3: Adaptive Versus Constant Underrelaxation Factors for XOR Example: Unsuccessful Trials

tive underrelaxation factors over Newton's method with constant underrelaxation factors in the number of singular matrix errors. Here, however, the pseudo-Newton method with constant underrelaxation factors outperforms all other methods. As with the neuron example, we observe a cost in using adaptive underrelaxation factors of an increase in trials resulting in convergence of the error function to a local minimum. Our results indicate further experiments and investigation are necessary to determine the usefulness, costs, and employability of our methods.

Chapter 5

An Algebraic Approach to the Initial Weight Problem

As we have seen in our previous experiments, a significant obstacle to the successful training of any artificial neural network is the existence of local minima of the error function. In particular, the complex one-step Newton steplength algorithm (Theorem 3.4.4) only guarantees convergence of a real-valued function with complex domain to a local minimum when using the Newton or pseudo-Newton method to minimize the function. A similar problem occurs with the typical gradient descent method, and in fact most minimization algorithms will only guarantee convergence to a local minimum. However, successful training of an artificial neural network requires that the real-valued error function be minimized to within a fixed tolerance of the global minimum of zero. It is evident in our experiments, and in fact more widely in the literature, that convergence to a local versus global minimum relies heavily on the choice of the initial weights. In fact, Newton's method is well-known to be particularly sensitive to the choice of the initial iterate, which leads to problems both with overshooting (see Theorem 3.4.4) as well as blow-ups and convergence to the local minimum closest to the initial iterate instead of a global minimum [38]. In what follows, we take an algebraic approach to address this initial weight problem for complex-valued polynomial neural networks.

We employ a constrained approach to narrow the domain from which to choose initial weights to aid in artificial neural network training. In training an artifi-

cial neural network, we desire to approximate a function of the input vectors that matches the training set. Because of this, it is important that the data be maintained throughout the training process. Typical backpropagation holds the training set constant at each step while updating the weights, relying on the entire training set to compute the error function. This process of utilizing the entire training set to train a network is sometimes referred to as batch training [15]. In incremental training, by contrast, a network is trained using only one data point at a time. Here the phenomenon of interference results in the network "forgetting" previously learned data, and preservation of the original data set becomes even more necessary. Some authors take the approach of choosing a constraint based on the original data set, then performing a constrained incremental training algorithm to train the network in order to preserve the original data set [13, 14]. We follow a similar idea here, however we continue to use batch training with the backpropagation algorithm. We choose constraints based on the original training set which will allow us to narrow the region from which we choose random initial weights and which guarantees existence of a global minimum of zero for the error function in that region. Since our focus is on complex-valued neural networks with polynomial activation functions, we can employ techniques from algebraic geometry to find such a region that satisfies our conditions.

In what follows, we first apply this approach to a complex-valued polynomial neuron for which we have a simpler formulation of the error function, and we develop a theoretical framework to aid in choosing a region from which to choose the initial weights based on algebraic methods. We propose a search strategy in Proposition 5.3.3 that allows us to choose a rectangular region from which we can randomly choose initial weights, and then we extend our results to complex-valued polynomial MLPs.

5.1 Translation of a Training Set into a Polynomial System

We consider again a complex-valued artificial neuron as shown in Figure 2.1 with m inputs $x_1, ..., x_m$, m weights $w_1, ..., w_m$, and output y defined by

$$y = g\left(\sum_{i=1}^{m} w_i x_i\right),$$

where $g \in \mathbb{C}[z]$ is a polynomial of degree n. Let

$$g(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0,$$

and suppose further that $a_0, a_1, ..., a_n \in \mathbb{R}$, so that $\overline{g(z)} = g(\overline{z})$. As we have previously seen in Chapter 4, this allows us to apply the Newton backpropagation algorithm (Theorem 3.2.1) to train the neuron. Recall that we typically train the neuron using a training set

$$T = \{(z_{t1}, ..., z_{tm}, d_t) \,|\, t = 1, ..., N\}$$

by minimizing the standard real-valued sum-of-squares error function

$$E = \frac{1}{N} \sum_{t=1}^{N} |y_t - d_t|^2. \tag{5.1.1}$$

Since the neuron has a polynomial activation function, we can instead take an algebraic rather than numerical approach to the problem of training the neuron. For $t = 1, ..., N$, the training point $(z_{t1}, ..., z_{tm}, d_t)$ represents a desired input-output pair that the neuron should replicate to within a desired tolerance once the neuron is trained. That is, the approximation

$$d_t \approx g\left(\sum_{i=1}^{m} w_i z_{ti}\right)$$

should hold for $t = 1, ..., N$ to within a specified tolerance once the final weights $w_1, ..., w_m$ are determined via the backpropagation algorithm. If the error function (5.1.1) is in fact minimized to zero, then for each training point, we have $y_t - d_t = 0$, that is,

$$d_t = y_t = g\left(\sum_{i=1}^{m} w_i z_{ti}\right).$$

So for a polynomial neuron, minimization of the error function to a global minimum value of zero is equivalent to finding the zeros of the polynomial system

$$\left\{ g\left(\sum_{i=1}^{m} w_i z_{ti}\right) - d_t = 0 \,|\, t = 1, ..., N \right\} \tag{5.1.2}$$

of N polynomial equations in the m weight vectors $w_1, ..., w_m$. Let $h_1, ..., h_N \in \mathbb{C}[w_1, ..., w_m]$, where

$$h_t(w_1, ..., w_m) := g\left(\sum_{i=1}^{m} w_1 z_{ti}\right) - d_t \tag{5.1.3}$$

is the polynomial corresponding to the tth training point for $t = 1, ..., N$. Then the system (5.1.2) can be rewritten as

$$h_1(w_1, ..., w_m) = \cdots = h_N(w_1, ..., w_m) = 0. \tag{5.1.4}$$

Thus, we can take the algebraic approach of training the neuron by finding the zeros of the system (5.1.4).

As we have done previously, let $w_j = w_{j1} + iw_{j2}$ where $w_{j1}, w_{j2} \in \mathbb{R}$ for $j = 1, ..., m$, and set $\mathbf{w} = (w_1, ..., w_m)$, $\mathbf{w}_1 = (w_{11}, ..., w_{m1})$, and $\mathbf{w}_2 = (w_{12}, ..., w_{m2})$ so that $\mathbf{w} = \mathbf{w}_1 + i\mathbf{w}_2$. For each training point $(z_{t1}, ..., z_{tm}, d_t)$ for $t = 1, ..., N$, let $z_{tj} = z_{tj1} + iz_{tj2}$ where $z_{tj1}, z_{tj2} \in \mathbb{R}$ for $j = 1, ..., m$, and $d_t = d_{t1} + id_{t2}$ where

$d_{t1}, d_{t2} \in \mathbb{R}$. Then for $t = 1, ..., N$, we can expand (5.1.3) and write

$$\begin{aligned} h_t(\mathbf{w}) &= h_t(w_1, ..., w_m) \\ &= g\left(\sum_{j=1}^{m}(w_{j1} + iw_{j2})(z_{tj1} + iz_{tj2})\right) - (d_{t1} + id_{t2}) \\ &= h_{t1}(w_{11}, ..., w_{m1}, w_{12}, ..., w_{m2}) + ih_{t2}(w_{11}, ..., w_{m1}, w_{12}, ..., w_{m2}) \\ &= h_{t1}(\mathbf{w}_1, \mathbf{w}_2) + ih_{t2}(\mathbf{w}_1, \mathbf{w}_2) \end{aligned}$$

where $h_{t1}, h_{t2} \in \mathbb{R}[\mathbf{w}_1, \mathbf{w}_2]$. Then $h_t(\mathbf{w}) = 0$ if and only if $h_{t1}(\mathbf{w}_1, \mathbf{w}_2) = 0$ and $h_{t2}(\mathbf{w}_1, \mathbf{w}_2) = 0$, so the complex system (5.1.4) can be rewritten as the real system

$$\{h_{t1}(\mathbf{w}_1, \mathbf{w}_2) = 0 \text{ and } h_{t2}(\mathbf{w}_1, \mathbf{w}_2) = 0 \,|\, t = 1, ..., N\} \tag{5.1.5}$$

of $2N$ polynomial equations in the $2m$ real variables $w_{11}, ..., w_{m1}, w_{12}, ..., w_{m2}$.

Before we develop a theoretical framework for root location for the complex neuron, we revisit our neuron example from Section 2 of Chapter 4. Recall that we train a neuron with $m = 2$ inputs and polynomial activation function

$$g(z) = \frac{1}{2} + \frac{1}{4}z - \frac{1}{48}z^3$$

using the training set given in Table 4.1. In our current setting, minimization of the error function

$$E = \frac{1}{2}\left(|y_1 - d_1|^2 + |y_2 - d_2|^2\right)$$

is equivalent to finding the zeros of the polynomial system

$$\begin{cases} h_1(w_1, w_2) := g(w_1 + 2w_2) - 1 = \dfrac{1}{2} + \dfrac{1}{4}(w_1 + 2w_2) - \dfrac{1}{48}(w_1 + 2w_2)^3 - 1 = 0, \\ \\ h_2(w_1, w_2) := g(-w_2) - 0 = \dfrac{1}{2} + \dfrac{1}{4}(-w_2) - \dfrac{1}{48}(-w_2)^3 - 1 = 0. \end{cases} \tag{5.1.6}$$

For purposes of illustration, we can explicitly solve (5.1.6) to find the nine solutions listed in Table 5.1. Figures 5.1 and 5.2 show rectangular regions in $\mathbb{C}^2$ taken from

w_1	w_2	**Color Coding**
4.208	-4.208	Teal
$10.519 - 1.130i$		Pink
$10.519 + 1.130i$		Orange
$-2.104 + 1.130i$	$2.104 - 1.130i$	Green
$-2.104 + 3.391i$		Yellow
$-8.415 + 2.261i$		Red
$-2.104 - 1.130i$	$2.104 + 1.130i$	Blue
$-2.104 - 3.391i$		Cyan
$-8.415 - 2.261i$		Purple

Table 5.1: Solutions of the polynomial system (5.1.6) and color coding

the planes $\mathbb{R} \times \mathbb{R}\, i$ and $\mathbb{R}\, i \times \mathbb{R}\, i$, respectively, which are color coded to represent the basins of attraction of choice of initial weights corresponding to each of the nine roots for the neuron trained using the pseudo-Newton method with complex one-step Newton steplengths. Note that all of the roots are not represented in these particular sections of the weight space. As in Figures 4.1 and 4.2, the graphs on the left correspond to the use of a constant underrelaxation factor $\omega = 0.5$, and the graphs on the right correspond to the use of the adaptive underrelaxation factor algorithm given by Corollary 4.1.2. White regions represent singular matrix errors, and grey regions represent convergence of the error function to a local minimum.

Note the similarity in the basins of attraction obtained when using a constant versus an adaptive underrelaxation factor. In the next section, we develop a method that will allow us to choose initial weights from regions entirely contained within the basins of attraction of the roots of our system. For this example, we note in particular that the typical choice of initial weights taken randomly from a region near the origin is likely to result in many singular matrix errors and local minima (see Figure 5.1(C)), so this is a region we wish to avoid. We see that the typical choice of initial weights is not always sufficient to successfully train a neuron (or, more generally, a network).

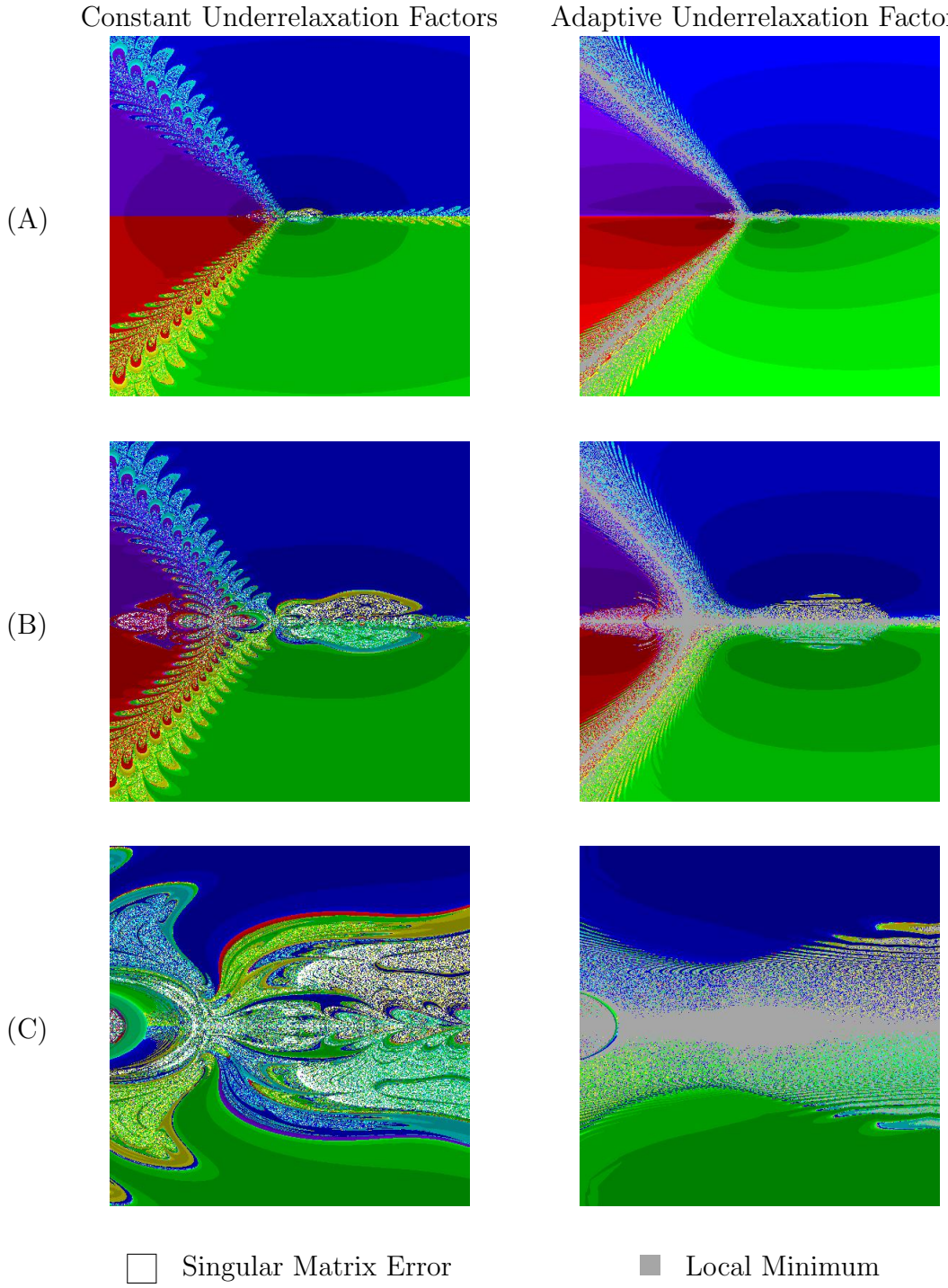


Figure 5.1: These graphs show the basins of attraction of the zeros of (5.1.6) given in Table 5.1 for initial weights (w_1, w_2) chosen from the regions (A) $[-20, 20] \times [-20, 20]\, i$, (B) $[-5, 5] \times [-5, 5]\, i$, and (C) $[-1, 1] \times [-1, 1]\, i$. The horizontal axis represents the choice of w_1 from the real line $\mathbb{R}$, and the vertical axis represents the choice of w_2 from the imaginary line $\mathbb{R}\, i$. There is a slight increase in area of the basins of attraction as well as a significant decrease in singular matrix errors when an adaptive versus constant underrelaxation factor is used.

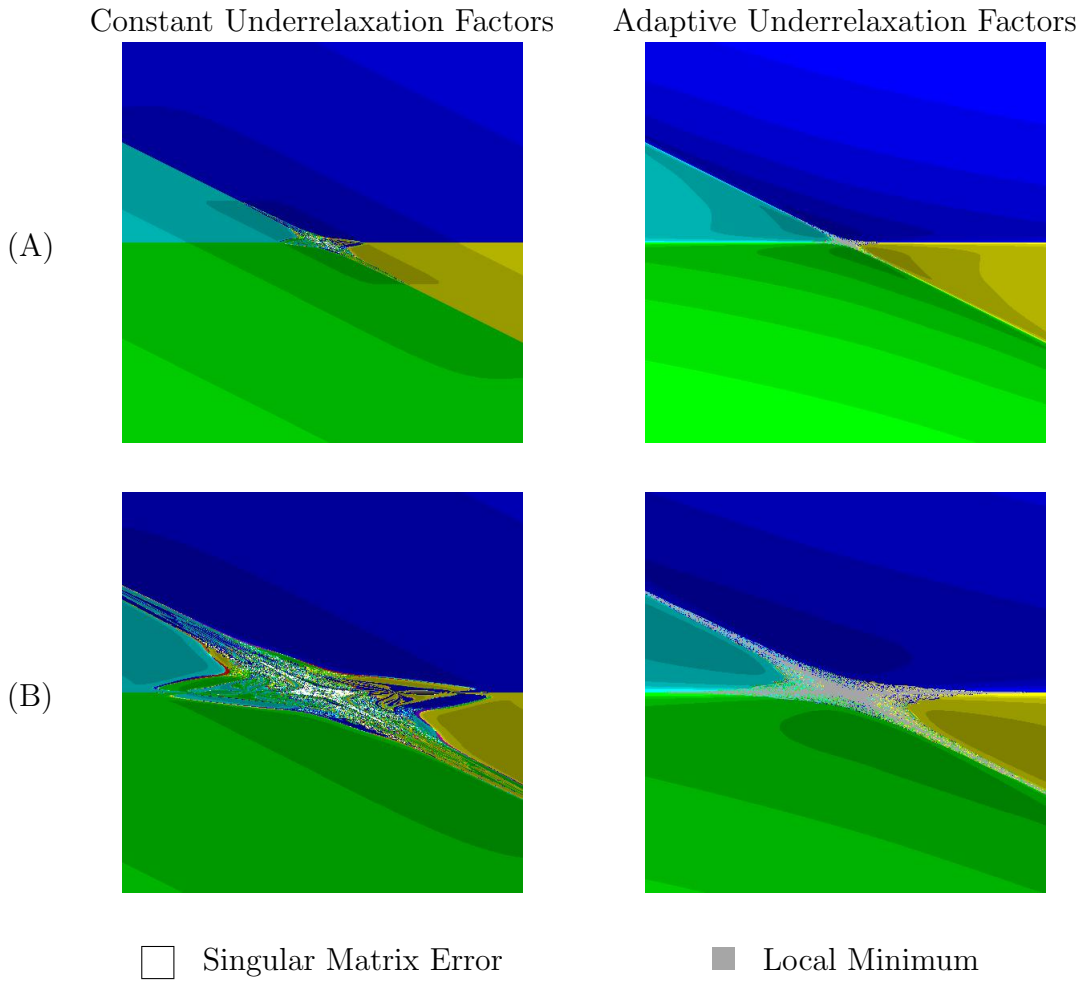


Figure 5.2: These graphs show the basins of attraction of the zeros of (5.1.6) given in Table 5.1 for initial weights (w_1, w_2) chosen from the regions (A) $[-20, 20]\, i \times [-20, 20]\, i$ and (B) $[-5, 5]\, i \times [-5, 5]\, i$. The horizontal axis represents the choice of w_1 from the real line $\mathbb{R}$, and the vertical axis represents the choice of w_2 from the imaginary line $\mathbb{R}\, i$. There is a slight increase in area of the basins of attraction as well as a significant decrease in singular matrix errors when an adaptive versus constant underrelaxation factor is used.

5.2 Real Root Location for a Neuron System

In training a complex-valued neuron or neural network, the initial weights are often chosen with the real and imaginary parts randomly chosen from the interval $[0, 1]$ or some other pre-chosen interval. However, it is typically not guaranteed that such randomly chosen initial weights lie "near" a global minimum of the error function. In this section, we describe the technique of real root isolation and location from [40] using the notation and exposition as outlined in [12] to choose a region $R \subseteq \mathbb{R}^{2m}$ that contains a zero of (5.1.5) which corresponds to a global minimum of the error function (5.1.1). We can then randomly choose the real and imaginary parts of initial weights from the region R and employ the Newton's method back propagation algorithm to the neuron in order to minimize the error function to this global minimum.

We begin by noting that, in fact, since $\mathbb{R}[\mathbf{w}_1, \mathbf{w}_2] \subseteq \mathbb{C}[\mathbf{w}_1, \mathbf{w}_2]$, the polynomials h_{t1}, h_{t2} for $t = 1, ..., N$ are in fact complex polynomials in the $2m$ complex variables $w_{11}, ..., w_{m1}, w_{12}, ..., w_{m2}$. As the variables $w_{11}, ..., w_{m1}, w_{12}, ..., w_{m2}$ actually represent the real and imaginary parts of the weight vectors $w_1, ..., w_m$, we are searching for the real roots of the complex system (5.1.5). Hence the technique of real root location and isolation we follow is appropriate in this setting. In what follows, we thus work in the polynomial ring $\mathbb{C}[\mathbf{w}_1, \mathbf{w}_2]$.

Let $I \subseteq \mathbb{C}[\mathbf{w}_1, \mathbf{w}_2]$ be the ideal generated by the polynomials h_{t1}, h_{t2} for $t = 1, ..., N$:

$$I := \langle h_{11}, h_{12}, ..., h_{N1}, h_{N2} \rangle.$$

Then the variety $V(I) \subseteq \mathbb{C}^{2m}$ is equal to the set of zeros of the complex system (5.1.5). We may assume from this point forward without loss of generality that $V(I)$ is finite as follows. Suppose $V(I)$ is infinite. We may then add constraint polynomials $\tilde{h_1}, ..., \tilde{h_r} \in \mathbb{R}[\mathbf{w}_1, \mathbf{w}_2]$ for some $r \in \mathbb{N}$ to define

$$\tilde{I} := \langle h_{11}, h_{12}, ..., h_{N1}, h_{N2}, \tilde{h_1}, ..., \tilde{h_r} \rangle$$

such that the variety $V(\tilde{I})$ is finite. Then $I \subseteq \tilde{I}$, so $V(\tilde{I}) \subseteq V(I)$ and thus points in $\mathbb{R}^{2m}$ that are zeros of the system

$$h_{11} = h_{12} = \cdots = h_{N1} = h_{N2} = \tilde{h_1} = \cdots = \tilde{h_r} = 0 \tag{5.2.1}$$

are also zeros of the original system (5.1.5). Since our neuron setup only requires us to find one root of the system (5.1.5) and not all the roots, it is sufficient to locate a root of the system (5.2.1). Hence in what follows we can replace I with $\tilde{I}$ to isolate and locate a real root of the system. Finiteness of the variety $V(I)$ guarantees that we can find a rectangle $R \subseteq \mathbb{R}^{2m}$ that contains no more than one of the points in $V(I)$.

Define the algebra A to be

$$A := \mathbb{C}[\mathbf{w}_1, \mathbf{w}_2]/I.$$

By the Finiteness Theorem (see Chapter 2, Section 2 of [12]), since $V(I)$ is a finite set, A is finite dimensional over $\mathbb{C}$. We wish to find a basis for A over $\mathbb{C}$. To this end, fix a natural ordering on the $2m$ variables $w_{11}, ..., w_{m1}, w_{12}, ..., w_{m2}$ as

$$w_{11} > ... > w_{m1} > w_{12} > ... > w_{m2}, \tag{5.2.2}$$

and choose a monomial order $<$ based off of (5.2.2). Let $\text{LT}(I)$ denote the set of leading terms of all the elements of I with respect to the monomial order $<$; that is,

$$\text{LT}(I) = \{\text{LT}(f) \,|\, f \in I\},$$

where $\text{LT}(f)$ denotes the leading term of the polynomial f with respect to $<$. Let $\alpha = (\alpha_{11}, ..., \alpha_{m1}, \alpha_{12}, ..., \alpha_{m2}) \in \mathbb{N}^{2m}$, and let

$$\mathbf{w}^\alpha = w_{11}^{\alpha_{11}} \cdots w_{m1}^{\alpha_{m1}} w_{12}^{\alpha_{12}} \cdots w_{m2}^{\alpha_{m2}}$$

denote the monomial in $w_{11}, ..., w_{m1}, w_{12}, ..., w_{m2}$ of multi-degree α. Then the set of monomials

$$B := \{\mathbf{w}^\alpha \,|\, \mathbf{w}^\alpha \notin \langle \text{LT}(I) \rangle\}$$

form a basis for A and this set is finite. Let $\dim_{\mathbb{C}} A = a$. Letting G denote a Groebner basis for I with respect to $<$ (so that $\langle \text{LT}(G) \rangle = \langle \text{LT}(I) \rangle$, the set

$$B := \{\mathbf{w}^{\alpha^{(i)}} \,|\, i = 1, ..., a \text{ and } \mathbf{w}^{\alpha^{(i)}} \notin \langle \text{LT}(G) \rangle\}$$

serves as a basis for A.

Let $p \in \mathbb{C}[\mathbf{w}_1, \mathbf{w}_2]$, and define the map $m_p : A \to A$ to be multiplication by p in the algebra A. If $\overline{q} \in A$ denotes the residue of $q \in \mathbb{C}[\mathbf{w}_1, \mathbf{w}_2]$ modulo I, then

$$m_p(\overline{q}) = \overline{pq} \in A.$$

The map $m_p : A \to A$ is a linear map [12]. We denote also by m_p the matrix of the linear map m_p with respect to B. Next, we define the symmetric bilinear form on A (see again [12]) $S : A \times A \to \mathbb{C}$ by

$$S(p, q) = \mathrm{Tr}(m_p m_q) = \mathrm{Tr}(m_{pq}) \tag{5.2.3}$$

for $p, q \in A$. For $f \in \mathbb{C}[\mathbf{w}_1, \mathbf{w}_2]$, let $S_f : A \times A \to \mathbb{C}$ be the bilinear form on A defined by

$$S_f(p, q) = S(fp, q) = \mathrm{Tr}(m_{fp} m_q) = \mathrm{Tr}(m_{fpq}) \tag{5.2.4}$$

for $p, q \in A$.

Let $\sigma(S_f)$ and $\rho(S_f)$ denote the signature and rank, respectively, of S_f with respect to the basis B of A. Using our notation, we state the following theorem.

Theorem 5.2.1 ([40], Theorem 2.1)**.** *Let $k \subseteq \mathbb{R}$ be a field and suppose $V(I)$ is a finite affine algebraic variety defined by $I = \langle h_{11}, ..., h_{m1}, h_{12}, ..., h_{m2} \rangle$, where $h_{jk} \in k[\mathbf{w}_1, \mathbf{w}_2]$ for $j = 1, ..., m$, $k = 1, 2$. Then, for $f \in k[\mathbf{w}_1, \mathbf{w}_2]$,*

$$\sigma(S_f) = \#\{p \in V(I) \cap \mathbb{R}^{2m} \mid f(p) > 0\} - \#\{p \in V(I) \cap \mathbb{R}^{2m} \mid f(p) < 0\}$$

and

$$\rho(S_f) = \#\{p \in V(I) \cap \mathbb{C}^{2m} \mid f(p) \neq 0\},$$

where σ denotes the signature and ρ denotes the rank of the associated bilinear form S_f.

For a given polynomial $f \in \mathbb{R}[\mathbf{w}_1, \mathbf{w}_2]$, Theorem 5.2.1 yields the following information about the finite variety $V(I)$ (again see [12]). Suppose $f \notin I$.

1. For the constant polynomial $1 \in \mathbb{R}[\mathbf{w}_1, \mathbf{w}_2]$, the bilinear form $S_1 : A \times A \to \mathbb{C}$

is defined by

$$S_1(p,q) = S(1p,q) = \mathrm{Tr}(m_{1p}m_q) = \mathrm{Tr}(m_{pq}) \tag{5.2.5}$$

for $p, q \in A$. The signature of this form is

$$\begin{aligned}\sigma(S_1) &= \#\{p \in V(I) \cap \mathbb{R}^{2m} \,|\, 1 > 0\} - \#\{p \in V(I) \cap \mathbb{R}^{2m} \,|\, 1 < 0\} \\ &= \#\{p \in V(I) \cap \mathbb{R}^{2m} \,|\, 1 > 0\} \\ &= \#\{p \in V(I) \cap \mathbb{R}^{2m}\},\end{aligned} \tag{5.2.6}$$

giving the number of real points in the variety $V(I)$. If $\sigma(S_1) = 0$, then the system (5.1.5) has no real solutions. Since the real system (5.1.5) is equivalent to the complex system (5.1.4), this implies that (5.1.4) has no solution, and thus that the associated error function (5.1.1) does not attain a global minimum of zero. Hence, the neuron is untrainable. So, to go forward with this approach, we require that $\sigma(S_1) > 0$.

2. The rank
$$\rho(S_f) = \#\{p \in V(I) \cap \mathbb{R}^{2m} \,|\, f(p) \neq 0\}$$
gives the number of points in the variety $V(I)$ at which the polynomial f does not vanish. Since $f \notin I$, we cannot have $V(I) \subseteq V(f)$, for if so, then
$$I = \mathfrak{I}(V(I)) \supseteq \mathfrak{I}(V(f)) = \langle f \rangle,$$
where $\mathfrak{I}(X)$ denotes the ideal of polynomials which vanish on the set $X \subseteq \mathbb{C}^{2m}$ [16]. Therefore $f \in I$, a contradiction. In particular, calculation of the rank $\rho(S_f)$ provides a check that, in fact, $f \notin I$, and we avoid some extra computations.

3. Let
$$n_f^+ := \#\{p \in V(I) \cap \mathbb{R}^{2m} \,|\, f(p) > 0\}$$
and
$$n_f^- := \#\{p \in V(I) \cap \mathbb{R}^{2m} \,|\, f(p) < 0\}$$

Then the signature of S_f can be written as

$$\sigma(S_f) = n_f^+ - n_f^-. \tag{5.2.7}$$

The signature of the form S_{f^2} is

$$\begin{aligned}\sigma(S_{f^2}) &= \#\{p \in V(I) \cap \mathbb{R}^{2m} \,|\, (f(p))^2 > 0\} - \#\{p \in V(I) \cap \mathbb{R}^{2m} \,|\, (f(p))^2 < 0\} \\ &= \#\{p \in V(I) \cap \mathbb{R}^{2m} \,|\, (f(p))^2 > 0\} \\ &= n_f^+ + n_f^-.\end{aligned} \tag{5.2.8}$$

Thus the two signatures $\sigma(S_f)$ and $\sigma(S_{f^2})$ give enough information to find the number of points in $V(I)$ at which f is positive and the number of points in $V(I)$ at which f is negative. In particular, combining (5.2.7) and (5.2.8), we have

$$n_f^- = \frac{\sigma(S_{f^2}) - \sigma(S_f)}{2}. \tag{5.2.9}$$

We employ the above algorithm in our setting to isolate a point of the variety $V(I)$ as outlined in the following section.

5.3 A Search Strategy for Isolation of a Real Root of a Polynomial System

We employ the calculations 1 – 3 in the previous section to develop a search strategy to isolate a point in the variety $V(I)$ in the following manner. Recall that without loss of generality, we may assume that $V(I)$ is finite via the addition of constraints given in (5.2.1) to the ideal I. We revisit this approach here to isolate a single root of the system (5.1.5). Assume that $V(I)$ contains at least one point (this may be checked, as in the previous section, by computing the signature $\sigma(S_1)$, where S_1 is the bilinear form defined by (5.2.5) on $A = \mathbb{C}[\mathbf{w}_1, \mathbf{w}_2]/I$). We now choose

constraints $l_1, ..., l_s \in \mathbb{R}[\mathbf{w}_1, \mathbf{w}_2]$ such that the system

$$h_{11} = h_{12} = \cdots = h_{m1} = h_{m2} = l_1 = \cdots l_s = 0 \tag{5.3.1}$$

has exactly one solution. In particular, such polynomials $l_1, ..., l_s$ exist, for if $P = (p_{11}, ..., p_{m1}, p_{12}, ..., p_{m2}) \in V(I)$, then if

$$l_{jk}(w_1, w_2) = l_{jk}(w_{11}, ..., w_{m1}, w_{12}, ..., w_{m2}) = w_{jk} - p_{jk}$$

for $j = 1, ..., m$ and $k = 1, 2$, then the system

$$h_{11} = h_{12} = \cdots = h_{m1} = h_{m2} = l_{11} = l_{12} = \cdots = l_{m1} = l_{m2} = 0$$

has exactly one solution, namely, P.

Remark 5.3.1. Note that we likely do not know the location of such a point P; in fact this is what we are trying to find. So here we just provide the existence of such constraint polynomials, and in general the polynomials $l_1, ..., l_s$ used will not have this form.

Remark 5.3.2. Define the ideal

$$J = \langle h_{11}, h_{12}, ..., h_{m1}, h_{m2}, l_1, ..., l_s \rangle. \tag{5.3.2}$$

We can determine the number of real points in $V(J)$ in the same way we determined the number of real points in $V(I)$ in the previous section, employing the following notation. Let $A^J = \mathbb{C}[\mathbf{w}_1, \mathbf{w}_2]/J$. Since $I \subseteq J$, $V(I) \supseteq V(J)$, so $V(J)$ is finite. By the Finiteness Theorem ([12]), A^J is finite dimensional, so given a basis B^J for A^J with respect to our order $<$, we may define the bilinear form $S^J : A^J \times A^J \to \mathbb{C}$ as in (5.2.3), and subsequently S^J_f as in (5.2.4) for an arbitrary polynomial $f \in \mathbb{R}[\mathbf{w}_1, \mathbf{w}_2]$. Then the signature $\sigma(S^J_1)$ computes the number of real points in the variety $V(J)$.

Once we have an ideal J such that $V(J)$ contains exactly one point, we wish to isolate a rectangular region $R \subseteq \mathbb{R}^{2m}$. For each w_{jk} with $j = 1, ..., m$ and $k = 1, 2$,

we can isolate a strip of width less than a prefixed $\epsilon > 0$ of the form

$$R_{jk} := \{(w_{11}, ..., w_{m1}, w_{12}, ..., w_{m2}) \,|\, a_{jk} \leq w_{jk} \leq b_{jk}\}$$

for some $a_{jk} \leq b_{jk}$ with $b_{jk} - a_{jk} < \epsilon$ that contains a point of $V(J)$ using a bisection method as outlined in Proposition 5.3.3. Then, since $V(J)$ contains exactly one point, the intersection

$$R = \bigcap_{k=1}^{2} \bigcap_{j=1}^{m} R_{jk} \tag{5.3.3}$$

must contain this point.

Proposition 5.3.3 (A Search Strategy for the Isolation of a Real Root of a Polynomial System in $\mathbb{R}^n$)**.** *Let $I = \langle g_1, ..., g_r \rangle \subseteq \mathbb{R}[x_1, ..., x_n] = \mathbb{R}[\mathbf{x}]$ be such that $V(I) \subseteq \mathbb{C}^n$ is finite and $V(I)$ contains at least one real solution. Let $x_1 > \cdots > x_n$ and fix a monomial order $<$ on $\mathbb{R}[x_1, ..., x_n]$.*

1. *Find polynomials $l_1, ..., l_s \in \mathbb{R}[\mathbf{x}]$ such that $V(J)$ contains exactly one point, where*

   $$J := \langle g_1, ..., g_r, l_1, ..., l_s \rangle.$$

   Set $A_J = \mathbb{C}[\mathbf{x}]/J$, and let B_J be a basis for A_J with respect to the order $<$. Define the bilinear form $S^J : A_J \times A_J \to \mathbb{C}$ by (5.2.3) and S_f^J by (5.2.4).

2. *Fix $\epsilon > 0$. For each variable x_i with $i = 1, ..., n$, we perform the following bisection algorithm.*

   (a) *Choose $a_i^{(0)} < b_i^{(0)}$ and define $f_i^{(0)} \in \mathbb{R}[\mathbf{x}]$ by*

   $$f_i^{(0)}(x_1, ..., x_n) = (x_i - a_i^{(0)})(x_i - b_i^{(0)}).$$

   Compute

   $$n^-_{f_i^{(0)}} = \frac{\sigma(S_{(f_i^{(0)})^2}) - \sigma(S_{f_i^{(0)}})}{2}.$$

   If $n^-_{f_i^{(0)}} = 0$, choose new values for $a_i^{(0)} < b_i^{(0)}$ and repeat step (a). If

$n^-_{f_i^{(0)}} = 1$, *the level set*

$$\{\mathbf{x} \in \mathbb{R}^n \mid f_i^{(0)} < 0\} = \{\mathbf{x} \in \mathbb{R}^n \mid a_i^{(0)} < x_i < b_i^{(0)}\}$$

contains a point in $V(J)$. *Continue on to step (b).*

(b) For $k \in \mathbb{N}$, *if* $b_i^{(k)} - a_i^{(k)} < \epsilon$, *set* $a_i = a_i^{(k)}$ *and* $b_i = b_i^{(k)}$ *and move to step 3. If* $b_i^{(k)} - a_i^{(k)} \geq \epsilon$, *define* $d_i^{(k)}, e_i^{(k)} \in \mathbb{R}[\mathbf{x}]$ *by*

$$d_i^{(k)}(x_1, ..., x_n) = (x_i - a_i^{(k)}) \left(x_i - \left(\frac{a_i^{(k)} + b_i^{(k)}}{2} \right) \right)$$

and

$$e_i^{(k)}(x_1, ..., x_n) = \left(x_i - \left(\frac{a_i^{(k)} + b_i^{(k)}}{2} \right) \right) (x_i - b_i^{(k)}).$$

Compute $n^-_{d_i^{(k)}}$ *and* $n^-_{e_i^{(k)}}$. *There are three possibilities.*

i. If $n^-_{d_i^{(k)}} = 1$, *then the strip*

$$\left\{ \mathbf{x} \in \mathbb{R}^n \;\middle|\; a_i^{(k)} < x_i < \frac{a_i^{(k)} + b_i^{(k)}}{2} \right\}$$

contains a point in $V(J)$. *Set* $a_i^{(k+1)} = a_i^{(k)}$ *and* $b_i^{(k+1)} = \frac{a_i^{(k)} + b_i^{(k)}}{2}$, *and repeat step (b).*

ii. If $n^-_{e_i^{(k)}} = 1$, *then the strip*

$$\left\{ \mathbf{x} \in \mathbb{R}^n \;\middle|\; \frac{a_i^{(k)} + b_i^{(k)}}{2} < x_i < b_i^{(k)} \right\}$$

contains a point in $V(J)$. *Set* $a_i^{(k+1)} = \frac{a_i^{(k)} + b_i^{(k)}}{2}$ *and* $b_i^{(k+1)} = b_i^{(k)}$ *and repeat step (b).*

iii. If both $n^-_{d_i^{(k)}} = 0$ *and* $n^-_{e_i^{(k)}} = 0$, *then a point in* $V(J)$ *lies on the hyperplane* $x_i - \left(\frac{a_i^{(k)} + b_i^{(k)}}{2} \right) = 0$. *Set* $a_i = b_i = \frac{a_i^{(k)} + b_i^{(k)}}{2}$ *and move to step 3.*

3. Set

$$R_i = \{\mathbf{x} \in \mathbb{R}^n \,|\, a_i < x_i < b_i\}.$$

and let

$$R = \bigcap_{i=1}^{n} R_i.$$

Then R contains the point in $V(J)$.

In searching for a rectangle containing a point of $V(I)$ in this manner by first reducing our system to one with exactly one solution, we avoid the complexities involved in checking multiple polynomial constraints when searching for one among multiple possible solutions to the original system [40]. Once we have obtained the rectangle R given by (5.3.3) of the desired width that contains the single point in $V(J)$, and by extension at least one point of V(I), we may then choose random initial weights $\mathbf{w}(0)$ from the rectangle R and apply the Newton or pseudo-Newton back propagation algorithm to train the network. Assuming R is contained in the path-connected component $L^0_{\mathbb{C}^{2m}}(E(\mathbf{w}(0)))$ of the level set $L_{\mathbb{C}^{2m}}(E(\mathbf{w}(0)))$ of the error function (see Equation (3.4.9)), and that $L^0_{\mathbb{C}^{2m}}(E(\mathbf{w}(0)))$ contains no other stationary points of the error function, Theorem 3.4.4 guarantees convergence to the global minimum. In practice we do not check each of these assumptions, but instead choose a small enough target width $\epsilon > 0$ so that each assumption will usually be satisfied.

5.4 A Polynomial System for the Multilayer Perceptron

We now extend our approach to choosing initial weights to the complex-valued polynomial multilayer perceptron as defined in Section 4 of Chapter 3 (see also Figure 2.2 for the network architecture). As opposed to the polynomial neuron case, minimization of the error function to a global minimum of zero corresponds to the solution of a system of polynomial equations corresponding to *each* training point. That is, if the error function is equal to zero, we must have $y_{tl} = d_{tl}$ for $l = 1, ..., C$ and $t = 1, ..., N$. So for the tth training point $(z_{t1}, ..., z_{tm}, d_{t1}, ..., d_{tC})$,

we have for each output node corresponding to $l = 1, ..., C$ the N equations

$$d_{tl} = g_L \left(\sum_{k=1}^{K_{L-1}} w_{lk}^{(L-1)} x_k^{(L-1)} \right),$$

$t = 1, ..., N$, where the $x_j^{(p)}$ are defined recursively as in (2.2), depending on the input $(z_{t1}, ..., z_{tm})$.

Let $\mathbf{w}^{(p-1)}$ represent the weight vector for the pth layer of the network as given in (2.3.1) for $p = 1, ..., L$. Since the activation function in each layer of the network is a polynomial, the tth training point corresponds to the C polynomial equations in the weight vectors $\mathbf{w}^{(0)}, ..., \mathbf{w}^{(L-1)}$, given by

$$h_{tl}(\mathbf{w}^{(0)}, ..., \mathbf{w}^{(L-1)}) := g_L \left(\sum_{k=1}^{K_{L-1}} w_{lk}^{(L-1)} x_k^{(L-1)} \right) - d_{tl} = 0$$

for $l = 1, ..., C$, where, again, the $x_j^{(p)}$ are defined recursively as in (2.2). Hence, minimizing the error function corresponds to solving the system of $C \cdot N$ equations

$$\{h_{tl}(\mathbf{w}^{(0)}, ..., \mathbf{w}^{(L-1)}) = 0 \,|\, t = 1, ..., N,\, l = 1, ..., C\} \tag{5.4.1}$$

in the complex variables $\mathbf{w}^{(0)}, ..., \mathbf{w}^{(L-1)}$. Note that in the pth layer of the network there are $K_p \cdot K_{p-1}$ weight vectors, so the polynomial ring $\mathbb{C}[\mathbf{w}^{(0)}, ..., \mathbf{w}^{(L-1)}]$ has $K := \sum_{p=1}^{L} K_p \cdot K_{p-1}$ variables.

To transform (5.4.1) into a real system, set $w_{ba}^{(p-1)} = w_{ba1}^{(p-1)} + i w_{ba2}^{(p-1)}$, where $w_{ba1}^{(p-1)}, w_{ba2}^{(p-1)} \in \mathbb{R}$ for $b = 1, ..., K_p$, $a = 1, ..., K_{p-1}$, and $p = 1, ..., L$, and let $\mathbf{w}^{(p-1)} = \mathbf{w}_1^{(p-1)} + i\mathbf{w}_2^{(p-1)}$. Then we can expand each polynomial h_{tl} into its real and imaginary parts via

$$\begin{aligned} h_{tl}(\mathbf{w}^{(0)}, ..., \mathbf{w}^{(L-1)}) = {} & h_{tl1}(\mathbf{w}_1^{(0)}, ..., \mathbf{w}_1^{(L-1)}, \mathbf{w}_2^{(0)}, ..., \mathbf{w}_2^{(L-1)}) \\ & + i h_{tl2}(\mathbf{w}_1^{(0)}, ..., \mathbf{w}_1^{(L-1)}, \mathbf{w}_2^{(0)}, ..., \mathbf{w}_2^{(L-1)}), \end{aligned}$$

where $h_{tl1}, h_{tl2} \in \mathbb{R}[\mathbf{w}_1^{(0)}, ..., \mathbf{w}_1^{(L-1)}, \mathbf{w}_2^{(0)}, ..., \mathbf{w}_2^{(L-1)}]$ for $t = 1, ..., N$ and $l = 1, ..., C$.

Then the complex polynomial system (5.4.1) can be rewritten as the real system

$$\{h_{tlk}(\mathbf{w}_1^{(0)}, ..., \mathbf{w}_1^{(L-1)}, \mathbf{w}_2^{(0)}, ..., \mathbf{w}_2^{(L-1)}) = 0 \,|\, t = 1, ..., N,\ l = 1, ..., C,\ k = 1, 2\}$$

of $2C \cdot N$ equations in the $2K$ real variables $\mathbf{w}_1^{(0)}, ..., \mathbf{w}_1^{(L-1)}, \mathbf{w}_2^{(0)}, ..., \mathbf{w}_2^{(L-1)}$. This allows us to apply Proposition 5.3.3 to find a rectangular box $R \subseteq \mathbb{R}^{2K}$ from which to choose initial weights for the Newton's method backpropagation algorithm.

Chapter 6

Conclusion and Future Work

Newton's method has been significantly under-utilized in the training of ANNs due to the computational inefficiency of computing and inverting the Hessian matrices. We have developed the backpropagation algorithm using Newton's method for complex-valued holomorphic multilayer perceptrons. The extension of RVNNs to CVNNs is natural and doing so allows the proper treatment of the phase information. However, the choice of nonlinear activation functions poses a challenge in the backpropagation algorithm. The usual complex counterparts of the commonly used real-valued activation functions are no longer unbounded: they have poles near zero, while other choices are not fully complex-valued functions. We proposed the use of holomorphic activation functions, which allowed for a simple formulation of the backpropagation algorithm using Newton's method akin to the typical gradient descent algorithm. We developed the complex one-step Newton step length algorithm to avoid the problem of overshooting of the iterates in using Newton's method to minimize real-valued complex functions. To provide experimental evidence for the choice of holomorphic functions as activation functions in addition to mathematical reasoning, we compared the results of using the complex-valued sigmoidal function as activation functions and the results of using its Taylor polynomial approximation as activation functions. Our experiments showed that when Newton's method was used for the XOR example, Taylor polynomial approximations are better choices. The use of complex one-step Newton steplengths further improved training iterations for the XOR example.

The use of polynomials as activation functions allows the possibility of rigorous

analysis of performance of the algorithm, as well as making connections with other topics of complex analysis, which are virtually nonexistent in complex-valued neural network studies so far. It also allows us to take an algebraic approach to the study of polynomial MLPs. Singularity of the Hessian matrices poses a significant obstacle to the use of Newton's method in the backpropagation algorithm, as we saw in the XOR examples. We developed an adaptive underrelaxation factor algorithm for minimization of real-valued complex functions via Newton's method that guarantees nonsingularity of the Hessian matrices. We applied our algorithm to an artificial neuron example as well as the XOR problem, finding that indeed the number of singular matrix errors was significantly decreased at a cost of increasing the frequency of the network being trained only to a local minimum. The reason behind the increase in local minimum training errors in using our algorithm is a problem for future work.

Newton's method is particularly sensitive to the choice of the initial iterate, and this was particularly evident in our XOR and neuron examples. We approached this problem from an algebraic viewpoint for polynomial MLPs and applied an algorithm for real root isolation and location of a polynomial system to the neuron and MLP cases. Further experiments in using this algorithm are a subject for future investigation.

Of particular interest for future work are the following questions and directions of study, which arise as extensions of this dissertation.

1. How can we improve our current algorithms, including the Newton and pseudo-Newton backpropagation algorithms as well as the complex one-step Newton steplength algorithm and the adaptive underrelaxation factor algorithm, in order to make them more computationally efficient and easier to implement in real-world applications?

2. How can we efficiently implement the search algorithms given in Chapter 5? In particular, can we develop an algebraic method to choose the constraint polynomials that are used to reduce the variety defined by a system of polynomial equations from an artificial neuron to one that contains exactly one point?

3. Further investigation of the algebraic properties of polynomial MLPs is warranted, as the use of polynomial activation functions for CVNNs opens up the

possibility of using additional theoretical techniques from algebraic geometry to study and improve training algorithms for these networks.

4. Some large-scale applications of our algorithms are necessary to show further experimental evidence of the superiority of our methods to the traditional gradient descent backpropagation algorithm. We plan to begin experiments using benchmark data sets from the UCI Machine Learning Repository to train polynomial MLPs using our algorithms. In particular, implementation of the methods in Chapter 5 is necessary in order to improve our search strategy.

Bibliography

[1] M.S. Al-Haik, H. Garmestani, and I.M. Navon. Truncated-newton training algorithm for neurocomputational viscoplastic model. *Computational methods in applied mechanics and engineering*, 192:2249–2267, 2003.

[2] Md. Faijul Amin, Md. Monirul Islam, and Kazuyuki Murase. Single-Layered Complex-Valued Neural Networks and Their Ensembles for Real-Valued Classification Problems. In *2008 International Joint Conference on Neural Networks*, pages 2500–2506. IEEE, 2008.

[3] Md. Faijul Amin, Md. Monirul Islam, and Kazuyuki Murase. Ensemble of single-layered complex-valued neural networks for classification tasks. *Neurocomputing*, 72:2227–2234, 2009.

[4] Md. Faijul Amin and Kazuyuki Murase. Single-layered complex-valued neural network for real-valued classification problems. *Neurocomputing*, 72:945–955, 2009.

[5] Md. Faijul Amin, Ramasamy Savitha, Muhammad Ilias Amin, and Kazuyuki Murase. Complex-Valued Functional Link Network Design by Orthogonal Least Squares Method for Function Approximation Problems. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1489–1496, July/August 2011.

[6] Md. Faijul Amin, Ramasamy Savitha, Muhammad Ilias Amin, and Kazuyuki Murase. Orthogonal least squares based complex-valued functional link network. *Neural Networks*, 32:257–266, 2012. 2012 Special Issue.

[7] Wee-Peng Ang and B. Farhang-Boroujeny. A New Class of Gradient Adaptive Step-Size LMS Algorithms. *IEEE Transactions on Signal Processing*, 49(4):805–810, April 2001.

[8] H.S.M. Beigi and C.J. Li. Learning Algorithms for Neural Networks Based on Quasi-Newton Methods With Self-Scaling. *Journal of Dynamical Systems, Measurement, and Control*, 115:38–43, March 1993.

[9] Sven Buchholz and Gerald Sommer. On Clifford neurons and Clifford multi-layer perceptrons. *Neural Networks*, 21:925–935, 2008.

[10] Kavita Burse, Anjana Pandey, and Ajay Somkuwar. Convergence Analysis of Complex Valued Multiplicative Neural Network for Various Activation Functions. In *2011 International Conference on Computational Intelligence and Communication Systems*, pages 279–282, 2011.

[11] John B. Conway. *Functions of One Complex Variable I.* Graduate Texts in Mathematics. Springer Science+Business Media, Inc., New York, 2 edition, 1978.

[12] David A. Cox, John Little, and Donal O'Shea. *Using Algebraic Geometry.* Springer Science+Business Media, Inc., New York, 2nd edition, 2005.

[13] Gianluca Di Muro and Silvia Ferrari. A Constrained-Optimization Approach to Training Neural Networks for Smooth Function Approximation and System Identificiation. In *2008 International Joint Conference on Neural Networks (IJCNN 2008)*, pages 2354–2360. IEEE, 2008.

[14] Silvia Ferreri and Mark Jensenius. A Constrained Optimazation Approach to Preserving Prior Knowledge During Incremental Training. *IEEE Transactions on Neural Networks*, 19(6):996–1009, June 2008.

[15] Silvia Ferreri and Robert F. Stengel. Smooth Function Approximation Using Neural Networks. *IEEE Transactions on Neural Networks*, 16(1):24–38, January 2005.

[16] William Fulton. *Algebraic Curves: An Introduction to Algebraic Geometry.* 28 January 2008. "http://www.math.lsa.umich.edu/∼wfulton/CurveBook.pdf".

[17] George M. Georgiou and Cris Koutsougeras. Complex Domain Backpropagation. *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, 39(5):300–334, May 1992.

[18] Su Lee Goh and Danilo P. Mandic. A Class of Gradient-Adaptive Step Size Algorithms for Complex-Valued Nonlinear Neural Adaptive Filters. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05)*, volume 5, pages V/253–V/256. IEEE, May 2005.

[19] F.V. Haeseler and H.O. Peitgen. Newton's Method and Complex Dynamical Systems. *Acta Applicandae Mathematicae*, 13:3–58, 1998.

[20] Martin T. Hagan and Mohammad B. Menhaj. Training Feedforward Networks with the Marquardt Algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993, November 1994.

[21] Andrew Ian Hanna and Danilo P. Mandic. A Fully Adaptive Normalized Nonlinear Gradient Descent Algorithm for Complex-Valued Nonlinear Adaptive Filters. *IEEE Transactions on Signal Processing*, 51(10):2540–2549, October 2003.

[22] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks*. The MIT Press, Cambridge, MA, 1995.

[23] Akira Hirose. Nature of complex number and complex-valued neural networks. *Frontiers of Electrical and Electronic Engineering in China*, 6(1):171–180, 2011.

[24] Akira Hirose. *Complex-Valued Neural Networks*, volume 400 of *Studies in Computational Intelligence*. Springer-Verlag Berlin Heidelberg, New York, 2nd edition, 2012.

[25] John H. Hubbard and Peter Papadopol. Newton's Method Applied to Two Quadratic Equations in $\mathbb{C}^2$ Viewed as a Global Dynamical System. *Memoirs of the American Mathematical Society*, 191(891), January 2008.

[26] Hamid A. Jalab and Rabha W. Ibrahim. New activation functions for complex-valued neural network. *International Journal of the Physical Sciences*, 6(7):1766–1772, April 2011.

[27] Taehwan Kim and Tülay Adali. Approximation by Fully Complex MLP Using Elementary Transcendental Functions. In *Neural Networks for Signal Processing XI, 2001. Proceedings of the 2001 IEEE Signal Processing Society Workshop*, pages 203–212. IEEE, 2001.

[28] Taehwan Kim and Tülay Adali. Fully Complex Multi-layer Perceptron Network for Nonlinear Signal Processing. *Journal of VLSI Signal Processing Systems*, 32(1/2):29–43, August-September 2002.

[29] Ken Kreutz-Delgado. The Complex Gradient Operator and the $\mathbb{CR}$-Calculus. University of California, San Diego, Version UCSD-ECE275CG-S2009v1.0, 25 June 2009. arXiv:0906.4835v1 [math.OC], June 2009.

[30] Diana Thomson La Corte and Yi Ming Zou. Newton's Method Backpropagation for Complex-Valued Holomorphic Multilayer Perceptrons. To appear in the International Joint Conference on Neural Networks (IJCNN 2014) Conference Proceedings.

[31] Yann Le Cun, Ido Kanter, and Sara A. Solla. Eigenvalues of Covariance Matrices: Application to Neural-Network Learning. *Physical Review Letters*, 66(18):2396–2399, May 1991.

[32] Henry Leung and Simon Haykin. The Complex Backpropagation Algorithm. *IEEE Transactions on Signal Processing*, 39(9):2101–2104, September 1991.

[33] Hualiang Li and Tülay Adali. Complex-Valued Adaptive Signal Processing Using Nonlinear Functions. *EURASIP Journal on Advances in Signal Processing*, 2008, 2008.

[34] Ming-Bin Li, Guang-Bin Huang, P. Saratchandran, and N. Sundararajan. Fully complex extreme learning machine. *Neurocomputing*, 68:306–314, October 2005.

[35] Jonathan H. Manton. Optimization Algorithms Exploiting Unitary Constraints. *IEEE Transactions on Signal Processing*, 50(3):635–650, March 2002.

[36] Indrajit Mukherjee and Srikanta Routroy. Comparing the performance of neural networks developed by using Levenberg–Marquardt and Quasi-Newton with the gradient descent algorithm for modelling a multiple response grinding process. *Expert Systems with Applications*, 39:2397–2407, February 2012.

[37] Iku Nemoto and Tomoshi Kono. Complex Neural Networks. *Systems and Computers in Japan*, 23(8):75–84, 1992. Translated from Denshi Joho Tsushin Gakkai Ronbunshi, Vol. 74-D-II, No. 9, pp. 1282-1288, September 1991.

[38] J.M. Ortega and W.C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, Inc., New York, NY, 1970.

[39] Anupama Pande and Vishik Goel. Complex-Valued Neural Network in Image Recognition: A Study on the Effectiveness of Radial Basis Function. *World Academy of Science, Engineering and Technology*, 26:220–225, 2007.

[40] P. Pedersen, M.-F. Roy, and A. Szpirglas. Counting real zeros in the multivariate case. In F. Eyssette et al, editor, *Computational Algebraic Geometry*, volume 109 of *Progress in Mathematics*, pages 203–224. Birkhäuser, Boston, 1993.

[41] Andrei D. Polyanin and Alexander V. Manzhirov. *Handbook of Mathematics for Engineers and Scientists*. Taylor & Francis Group, LLC, Boca Raton, FL, 2007.

[42] R. Savitha and S. Suresh and N. Sundararajan and P. Saratchandran. A new learning algorithm with logarithmic performance index for complex-valued neural networks. *Neurocomputing*, 72:3771–3781, 2009.

[43] Michael Reed and Barry Simon. *Methods of Modern Mathematical Physics I: Functional Analysis*. Academic Press, London, 1980.

[44] Reinhold Remmert. *Theory of Complex Functions*. Springer-Verlag, New York, NY, 1991.

[45] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning Internal Representations by Error Propagation. In D.E. Rumelhart and J.L. McCelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, chapter 8. Foundations M.I.T. Press, Cambridge, MA, 1986.

[46] R. Savitha, S. Suresh, N. Sundararajan, and H.J. Kim. Fast Learning Fully Complex-Valued Classifiers for Real-Valued Classification Problems. In D. Liu et al, editor, *Advances in Neural Networks–ISNN 2011, Part I*, volume 6675 of *Lecture Notes in Computer Science*, pages 602–609. Springer-Verlag Berlin Heidelberg, 2011.

[47] Laurent Sorber, Marc Van Barel, and Lieven De Lathauwer. Unconstrained optimization of real functions in complex variables. Technical Report TW592, Katholieke Universiteit Leuven, Heverlee (Belgium), April 2011.

[48] Hao Yu and Bogdan M. Wilamowski. Levenberg-Marquardt Training. In *Industrial Electronics Handbook, Vol. 5: Intelligent Systems*, chapter 12, pages 12–1 – 12–15. CRC Press, 2 edition, 2011.

[49] Hans Georg Zimmermann, Alexey Minin, and Victoria Kusherbaeva. Comparison of the Complex Valued and Ral Valued Neural Networks Trained with Gradient Descent and Random Search Algorithms. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 213–218, Bruges (Belgium), April 2011.

CURRICULUM VITAE

Diana Thomson La Corte

Education

Ph.D., Mathematics, University of Wisconsin-Milwaukee, August 2014

Advisor: Dr. Yi Ming Zou

Dissertation: Newton's Method Backpropagation for Complex-Valued Holomorphic Neural Networks: Algebraic and Analytic Properties

Areas of Study: Algebra and Applied Math

M.S., Mathematics, University of Wisconsin-Milwaukee, May 2010

B.S., Mathematics, University of Wisconsin-Milwaukee, May 2007

Employment

University of Wisconsin-Milwaukee

Graduate Teaching Assistant: As Instructor of Record

Calculus and Analytic Geometry III, Spring 2014, Fall 2013

Calculus and Analytic Geometry II, Fall 2012, Fall 2011, Fall 2010, Summer 2010, Spring 2010, Summer 2009

College Algebra, Fall 2009

Intermediate Algebra, Spring 2009

Graduate Teaching Assistant: As Teaching Assistant

Survey in Calculus and Analytic Geometry, Fall 2008

University of Wisconsin-Madison

Graduate Teaching Assistant: As Teaching Assistant

Survey in Calculus and Analytic Geometry: Fall 2008

University of Wisconsin-Milwaukee

Mathematics Tutor, 2005-2007

Papers

"Newton's Method Backpropagation for Complex-Valued Holomorphic Multilayer Perceptrons," with Yi Ming Zou. To appear in the International Joint Conference on Neural Networks (IJCNN 2014) Conference Proceedings.

Presentations

"Newton's Method Backpropagation for Holomorphic Complex-Valued Neural Networks." Talk given at the Applied and Computational Mathematics Seminar, University of Wisconsin-Milwaukee, February 2014.

"The Newton's Method Backpropagation Algorithm for Holomorphic Complex-Valued Neural Networks." Talk presented at the AMS Session on Statistical Modeling, Big Data, and Computing at the Joint Mathematics Meetings, Baltimore, Maryland, January 2014.

"The Newton's Method Backpropagation Algorithm for Complex-Valued Neural Networks." Talk given at the Classification Society 2013 Annual Meetings, University of Wisconsin-Milwaukee, June 2013.

Honors and Awards

University of Wisconsin-Milwaukee Morris and Miriam Marden Graduate Award, 2013. Award given for a mathematical paper of high quality. For "Newton's Method Backpropagation for Complex-Valued Holomorphic Multilayer Perceptrons."

University of Wisconsin-Milwaukee GAANN Fellowship, 2010-2013.

University of Wisconsin-Milwaukee Chancellor's Fellowship, 2008-2010 and 2013-2014.

University of Wisconsin-Milwaukee Alice Siu-Fun Leung Award, 2005, 2006, and 2007. Award given to outstanding undergraduate students in mathematical sciences.

National Merit Finalist, 2003.

Membership

American Mathematical Society, 2012-Present.