An artificial neural network method for solving boundary value problems with arbitrary irregular boundaries

1 author:

Kevin McFall
Kennesaw State University
37 PUBLICATIONS **373** CITATIONS

SEE PROFILE

AN ARTIFICIAL NEURAL NETWORK METHOD FOR SOLVING BOUNDARY VALUE PROBLEMS WITH ARBITRARY IRREGULAR BOUNDARIES

A Dissertation
Presented to
The Academic Faculty

By

Kevin S. McFall

In Partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy in Mechanical Engineering

Georgia Institute of Technology

May, 2006

AN ARTIFICIAL NEURAL NETWORK METHOD FOR SOLVING BOUNDARY VALUE PROBLEMS WITH ARBITRARY IRREGULAR BOUNDARIES

Approved by:

Dr. J. Robert Mahan, Advisor
Woodruff School of Mechanical Engineering
Georgia Institute of Technology

Dr. George Vachtsevanos
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Nader Sadegh
Woodruff School of Mechanical Engineering
Georgia Institute of Technology

Dr. Zhuomin Zhang
Woodruff School of Mechanical Engineering
Georgia Institute of Technology

Dr. Ali Siadat
Ecole Nationale Supérieure d'Arts et Métiers, Metz France

Date approved: April 5, 2006

Acknowledgements

This dissertation would never have come about without the support and direction of Dr. J. Robert Mahan. No one could hope for a more engaged advisor and better role model. The author owes a debt of gratitude to the *Conseil Régional de Lorraine* for its generous financial support of this research at the European Campus of the Georgia Institute of Technology, located in Metz, France. And although never directly involved in this work, my family – including a certain dangerous redhead – has perhaps made the largest contribution through years of constant and unwavering love and support.

Table of Contents

List of Tables

List of Figures

List of abbreviations

Term	Abbreviation
AI	Artificial intelligence
ANN	Artificial neural network
BC	Boundary condition
BVP	Boundary value problem
DE	Differential equation
FEM	Finite element method
MLP	Multi-layer perceptron
RBF	Radial basis function
RMS	Root mean squared
RPROP	A gradient descent learning algorithm
TAS	Trial approximate solution

Nomenclature

Quantity	Explanation
a	The index of one segment bordering the i^{th} segment, also a point in the domain used to define the length factor
a, a_1, a_2	Arbitrary quantities whose absolute value is of importance, the 1 subscript indicates the quantity before a training iteration and the 2 subscript after
a_i	Intercept of linear boundary segment i
$A(\mathbf{x})$	Returns correct BCs for every $\mathbf{x}$ along the boundary
$A_D(\mathbf{x})$	Component of $A(\mathbf{x})$ satisfying any Dirichlet conditions
$A_M(\mathbf{x})$	The original strategy was for A_M to only be a function of $\mathbf{x}$
$A_M(\mathbf{x}, N)$	Component of $A(\mathbf{x})$ satisfying any Neuman conditions
b	The bias for a node, also the index of one segment bordering the i^{th} segment, also a point in the domain used to define the length factor
b, b_1, b_2	An arbitrary quantity whose absolute value is of importance, the 1 subscript indicates the quantity before a training iteration and the 2 subscript after
$b_i(x)$	Function defining the shape of the i^{th} boundary segment for 2-D domains.
$B_{ij}(\mathbf{x})$	Function used to define $R(\mathbf{x})$
c_i	Dilation factor when generating curve C_i from curve C
$C_{ij}(\mathbf{x})$	Function used to define $R(\mathbf{x})$
C	Curve defined by a boundary segment shape
C_i	Curve generated by rotating, dilating, and translating C, $1 \le i \le 4$
d	Exponent on Dirichlet length factors in A_D, also a point in the domain used to define the length factor
D	Number of boundary segments with Dirichlet conditions
$\mathbf{D}'$	Vector difference between gradients of the true and TASs
D'_x	Difference in first partial derivatives of ψ and ψ_t with respect to x
D'_y	Difference in first partial derivatives of ψ and ψ_t with respect to y
$\mathbf{D}''$	Analogue to $\mathbf{D}'$ but for second derivatives
D''_x	Difference in second partial derivatives of ψ and ψ_t with respect to x
D''_y	Difference in second partial derivatives of ψ and ψ_t with respect to y
$\tilde{D}$	An arbitrary differential operator
e	A point in the domain used to define the length factor
E	Error function used to optimize the ANN parameters in θ
E_{BC}	Component of E associated with the BCs
E_{DE}	Component of E associated with the DE
E_{norm}	Error norm comparing TAS with exact solution
$f(\mathbf{x})$	An arbitrary function of $\mathbf{x}$
$F(\mathbf{x}, N)$	A function which contributes nothing to ψ_t along all boundaries

$g(\mathbf{x})$	An arbitrary function of $\mathbf{x}$
$\mathbf{g}$	Gradient vector of the error function E with respect to ANN weights θ
$g_i(\mathbf{x})$	Function satisfying either Dirichlet or Neuman BCs along the i^{th} boundary
$g_i'(\mathbf{x}, N)$	Function to aid A_M in satisfying a Neuman conditions on the i^{th} boundary
G	Signed error measure to be reduced during training
$\mathbf{G}$	Vector composed of the G values evaluated at each point in T
$g_D(\mathbf{x}_i)$	Function returning the Dirichlet boundary value at a given $\mathbf{x}_i$
$g_M(\mathbf{x}_i)$	Function returning the Neuman boundary slope at a given $\mathbf{x}_i$
$h(\mathbf{x}, N)$	Component of $A_M(\mathbf{x}, N)$ function
$h_1(\mathbf{x}, N)$	Candidate function for $h(\mathbf{x}, N)$
$h_2(\mathbf{x}, N)$	Candidate function composing $h(\mathbf{x}, N)$
H	The number of hidden nodes in the ANN
$\mathbf{H}$	Hessian matrix of the error function E with respect to ANN weights θ
i	A counter variable, also the imaginary unit $\sqrt{-1}$
$\hat{\mathbf{i}}$	Unit vector in the first cardinal direction
j	A counter variable
$\hat{\mathbf{j}}$	Unit vector in the second cardinal direction
J	The number of components in $\mathbf{x}$, which is the number of inputs to the ANN
$\mathbf{J}$	Jacobian matrix of the error function E with respect to ANN weights θ
k	A counter variable
l	A counter variable
L	A length factor
L_{basic}	Basic length factor considering the square hole as one segment
L_{circle}	Perpendicular distance length factor for the circular boundary
L_{comp}	Composite length factor for the square hole using the standard length factor
L_i	Length factor for the i^{th} boundary segment
L_{map}	Extension of L_{basic} using a complex mapping
m	Slope of a line, also an exponent on Neuman length factors in A_D
m_i	Slope of linear boundary segment i
M	Number of boundary segments with Neuman conditions
N	Output of the ANN
n	Grid size $n \times n$ (in 2-D) serving as the basis for determining membership in T
n_{ij}	The component in the j^{th} direction of the unit normal to the i^{th} boundary
$\hat{\mathbf{n}}_i$	The inwardly-directed unit normal to a given boundary evaluated at $\mathbf{x}_i$
$P_i(\mathbf{x})$	Function appearing in the general form of $g_i'(\mathbf{x}, N)$
P_i	Simplifying factor involving w_{ij} and λ_i
r	Radius for a circle
$R(\mathbf{x}, N)$	Function appearing in the general form of $g_i'(\mathbf{x}, N)$

$R_i(\mathbf{x})$	Function returning unity on segment i and zero on all other segments
s	Parameter representing the location along a boundary
s_0	A particular location on a segment boundary
S	Set of points within the DE domain for evaluating E_{DE}
S_D	Set of points along the boundary for evaluating E_{BC} for Dirichlet conditions
S_M	Set of points along the boundary for evaluating E_{BC} for Neuman conditions
$S_{ab}^{C_i}$	Unsigned line integral from points a to b along curve C_i
T	Set of points used for calculating either ANN error function or error norm
u_i	The bias for the i^{th} hidden node
$U(\mathbf{x})$	An arbitrary function of $\mathbf{x}$
v_i	The weight of the output node associated with the i^{th} hidden node
w_i	The i^{th} weight for a given node
w_{ij}	The weight connecting the j^{th} input to the i^{th} hidden node
W	The number of weights in the vector $\boldsymbol{\theta}$
x	The first component of the $\mathbf{x}$ vector
x_0	One coordinate of a particular point, usually on a boundary
x_1	One coordinate of a particular point
x_i or x_j	The i^{th} or j^{th} input to a node, also a component in the vector $\mathbf{x}$
$\mathbf{x}$	The vector containing the DE's independent variables, also the ANN input
y	The second component of the $\mathbf{x}$ vector
y_0	One coordinate of a particular point, usually on a boundary
y_1	One coordinate of a particular point
z	The biased and weighted sum of a node's inputs, also a complex number
z_i	The biased and weighted sum the inputs to the i^{th} hidden node
α	Scaling factor on the length factor
α_3, α_4	Factors in the length factor for suppressing discontinuities
β	Threshold on length factor value for determination of membership in T
δ	Symbol for a term approaching zero in a limit
δa	An arbitrarily small change in a due to an ANN training step
δb	An arbitrarily small change in b due to an ANN training step
γ	Exponent in the general form of $g_i'(\mathbf{x}, N)$, also a threshold on $\lvert G \rvert$ for determination of membership in T
Δ_i	The update-value for the i^{th} parameter in the vector $\boldsymbol{\theta}$
$\Delta_i^{(t)}$	The update-value for the i^{th} parameter in the vector $\boldsymbol{\theta}$ during the t^{th} iteration
η	Learning rate for weight updating
η^+	The factor greater than unity for increasing update-values Δ_i
η^-	The factor between zero and unity for decreasing update-values Δ_i
λ	Parameter in the Levenberg-Marquardt learning algorithm
λ_i	The order of the partial derivative with respect to x_i
μ	Exponent in the general form of $g_i'(\mathbf{x}, N)$
v_{ij}	Coefficient used in early attempts at defining the TAS
Λ	Total number of partial derivatives with respect to the components of $\mathbf{x}$

$\boldsymbol{\theta}$	The parameter, or weight, vector of an ANN
$\boldsymbol{\theta}_0$	A starting value for the ANN weight vector
$\boldsymbol{\theta}_i$	A value for the ANN weight vector during the i^{th} training iteration
$\boldsymbol{\theta}^*$	A value for the ANN weight vector at the error minimum
θ_i	The i^{th} weight in the vector $\boldsymbol{\theta}$
$\theta_i^{(t)}$	The i^{th} weight in the vector $\boldsymbol{\theta}$ during the t^{th} iteration
$\sigma(z)$	The logarithmic sigmoid function
$\sigma_i^{(\Lambda)}$	Shorthand notation for the Λ^{th} derivative of σ evaluated at z_i
$\sigma' \quad \sigma'' \quad \sigma'''$	The first three derivatives of the logarithmic sigmoid function
υ_{ij}	Coefficient used in early attempts at defining the TAS
ξ	Scaling factor on the perpendicular distance component of length factors
ψ	The exact solution to the DE defined by G
ψ_1	Simpler choice for ψ_t only involving the ANN output
ψ_2	Choice for form of ψ_t for automatic satisfaction of BCs
ψ_t	Approximate (also called trial) solution of the DE optimized by the ANN
ζ	Scaling factor in the denominator of A_M

Summary

The research behind the effort presented in this dissertation began by selecting numerical solution of boundary value problems (BVPs) as a vehicle for evaluating the benefits of weight reuse in multilayer perceptron artificial neural networks. The motivation was to build an intelligent system which incorporated previously obtained knowledge in order to be faster and more accurate in completing its tasks.

Research delving into solving BVPs with ANNs revealed that even though the topic has been studied for over ten years, it is still far from mature. The success of powerful software packages such as FEMLAB®, which utilize the finite element method (FEM) for solving BVPs, seems to have has stifled development of a competing method with many benefits over the FEM. The emphasis of the research documented in this dissertation shifted from weight reuse to further developing methods for solving BVPs using ANNs.

While the details of the research shifted, its spirit did not. The focus remained on integrating a priori knowledge into the system in order to make it more efficient. The result is a method which applies information in the form of the known boundary conditions (BCs) and the specified domain geometry. A method was developed which automatically satisfies all BCs, Dirichlet and/or Neuman, on any irregular domain boundary. A length factor was formulated which quantifies a “distance” from the domain boundary. It is used to ensure that BCs are satisfied on the boundary while providing a first approximation of the solution at points within the domain.

The approximate solution defined in this manner has exactly zero error on Dirichlet boundaries, something not available with the FEM. The ANN begins training with this reasonable, and sometimes impressively accurate, approximate solution rather than the random starting point typical of artificial intelligence optimization algorithms. The method thus begins at a location in the search space much closer to an optimal solution, leading to a method significantly simpler and often more accurate than other ANN methods for solving BVPs.

One benefit of the proposed method over the FEM, as noted previously, is that all BCs are automatically and exactly satisfied. The ANN is trained by reducing error in the given differential equation (DE) at certain points within the domain. Although defined by the user, selection of these points is significantly simpler than the often difficult definition of meshes for the FEM. The resulting approximate solution is continuous and differentiable, and can be evaluated at any location in the domain independent of the set of points used for training. This continuous solution eliminates the interpolation required of the discrete solution produced by the FEM.

Two often-voiced criticisms of ANN optimization are its data-driven black-box nature and the lack of solution error bounds. The proposed method addresses the first criticism by making use of BC and geometrical information to infuse knowledge into the approximate solution. Training begins then, not at a random location in the search space, but rather at a location benefiting from a priori knowledge of the problem. Defining an error bound for ANNs solving BVPs is especially difficult because the approximate solution is trained to improve upon satisfaction of the DE rather than reducing residual error in the solution itself. A theorem is presented however, which guarantees that

residual error in the solution will decrease at all locations in the domain so long as error in the DE is similarly reduced during training. This theorem is valid under certain circumstances which are reasonably approximated during actual training of the ANN.

The results of this dissertation offer a significant contribution to the field by successfully developing an ANN method for solving BVPs where all BCs are automatically satisfied for arbitrary irregular domains. It had already been established in the literature that such automatic BC satisfaction was beneficial when solving problems on rectangular domains, but this dissertation presents the first method applying the technique to irregular domain shapes. This was accomplished by developing an innovative length factor. This length factor ensured BC satisfaction on domain boundaries while providing a first-order approximation of the solution within the domain as well. The length factors extrapolate the values at Dirichlet boundaries into the domain, providing a solid starting point for ANN training to begin. The resulting method has been successful at solving even nonlinear and non-homogenous BVPs to accuracy sufficient for typical engineering applications.

1 Introduction

Artificial intelligence (AI) is a broad field including such topics as expert systems [1], fuzzy logic [2], artificial neural networks [3], evolutionary computing [4], and data mining [5]. Each of these various activities strives to enable a computer to complete tasks which normally require human intelligence. The classic definition of AI from the Turing test [6] requires simply that a human observer fail to distinguish between artificial and human responses to the same task. For instance, passengers would be hard-pressed to differentiate between human pilots and autopilots in a commercial airliner. A so-called intelligent system as defined by the Turing test is relatively simple to develop as long as the operating domain is kept small; the human pilot must take control when, such as during inclement weather, the operating conditions stray outside the bounds that the artificial system was developed to handle.

The Turing test requires only that an artificial system *behave* like a human, and not that it *think* like a human. Many believe that the latter must be true as well for a system to truly be intelligent. Artificial neural networks (ANNs) are based on biological neural networks composed of interconnected neurons which communicate with each other by transmitting binary electrical pulses. Biological neural networks contain billions of such neurons which act in parallel with an individual switching speed of approximately a millisecond, where a larger brain generally signifies higher intelligence. ANNs are based on this model but have at most hundreds of neurons operating with continuous signals in series at speeds in the nanosecond range, where the principle of Occam's Razor dictates that the smallest network capable of solving the problem provides the best solution. Critics

of ANNs maintain that they hardly resemble their biological counterparts: not only do they not *think* like a human, but they are black boxes whose inner workings are nearly impossible to interpret even when they do produce reasonable responses. ANNs were conceived before significant computing power was available [7, 8] but research did not begin to blossom [9-13] until the advent of the personal computer after which they have been applied to vast numbers of engineering problems [3].

Multi-layer feedforward ANNs are universal approximators [14] and thus, at least in theory, can generate a solution to any boundary value problem (BVP) with an arbitrarily small margin of error. The ANN practitioner quickly discovers that finding the correct network weight values producing the desired ANN output is not trivial for even moderately complex problems. ANNs are completely data-driven in the sense that they require no knowledge of the physics of the problem at hand. Research has led to the development of ANNs that solve problems without knowledge of the underlying physics by developing a wide array of tools such as advanced learning algorithms [15, 16], efficient feature extraction [17], network ensembles [18], and adaptive network architecture [19], among many others. These techniques enable ANNs to solve complicated problems, but add complexity to implementation of the solution which is likely outside the experience of the casual ANN user. Incorporating knowledge of the problem, such as satisfaction of boundary condition (BC) requirements using judiciously defined length factors, into the solution reduces the complexity of the problem and reduces or eliminates altogether the need for the advanced ANN tools. The method proposed in this dissertation for solving BVPs is significantly simpler, and often more accurate, than other ANN methods [20-23].

The proposed method is intended as an alternative to traditional numerical methods such as finite difference [24], finite element [25], and boundary element methods [26]. Certainly, the expectation that any ANN approach would ever replace commercially available finite element method software packages offering user-friendly interfaces for solving wide ranges of problems is likely not realistic. However, this method could be of use for some users who desire a flexible and relatively simple environment for solving BVPs without requiring intricate knowledge of the underlying theory behind the finite element method. It also avoids the complicated and arcane meshing strategies inherent in the finite element method. The proposed method is intentionally designed to be simple and intuitive so that it is accessible to users with minimal experience with multi-layer feedforward ANNs. Additionally, this ANN method provides several advantages over traditional numerical techniques: it is mesh-free, produces a continuous solution which can be evaluated at any point within the domain, enjoys superior interpolation ability, and satisfies the BCs, Dirichlet and/or Neuman, exactly everywhere along all arbitrarily shaped boundaries. Finally, the thrust of this doctoral effort has been more in the direction of advancing knowledge of AI pathology than in the direction of commercial product development.

1.1 Supervised ANNs solving BVPs as opposed to other problems

Supervised training of ANNs for solving boundary value problems is somewhat different than for solving other problems. First, boundary value problems are constrained optimization problems due to the requirement imposed by the BCs. This constraint is most conveniently handled by casting the trial approximate solution (TAS) in such a manner that the BCs are automatically satisfied [27-29].

The second difference is that only points on Dirichlet boundaries offer data where the true solution is known. ANNs trained to minimize error in the solution using these data would produce extremely poor results since points on Dirichlet boundaries certainly cannot characterize the behavior of the solution everywhere in the domain. Despite this obvious shortcoming, solution of BVPs benefits from the fact that the true solution must satisfy the DE everywhere within the domain. The ANN is thus trained to reduce the error in the DE itself rather than the error in its solution.

To clarify the distinction between errors in the solution and the DE, consider solving the DE given by

$$\tilde{D}\psi(\mathbf{x}) = f(\mathbf{x}) \tag{1}$$

subject to certains BCs where $\tilde{D}$ is some differential operator. The function $\psi(\mathbf{x})$ denotes the exact solution and $\mathbf{x} \in \Re^J$ is a J dimensional real vector in Euclidean space. The ANN output N will be used to generate a continuous trial solution $\psi_t(\mathbf{x},N)$ which approximates ψ. The TAS ψ_t can be evaluated at any $\mathbf{x}$ in the domain and compared to the true solution to quantify the error in the solution

$$D \equiv \psi(\mathbf{x}) - \psi_t(\mathbf{x}, N) \tag{2}$$

On the other hand, error in the DE is quantified by $\tilde{D}\psi_t(\mathbf{x}) - f(\mathbf{x})$, where the differential operator $\tilde{D}$ is applied to the trial approximate solution as opposed to the true solution. Error in the DE is the only metric available to evaluate the fitness of the TAS since the true solution, and thus D, is not known.

Obtaining input/output value pairs for solving BVPs with ANNs becomes trivial since the DE error function is everywhere an explicit function of the input values.

Generalization is no longer a vexing issue since unlimited training data can be generated anywhere in the input space with little computational effort. However, ensuring development of an accurate TAS is complicated by the fact that the approximate solution is optimized through the indirect measure of error in the DE. This work presents a theorem which guarantees that the error in the solution will converge to zero as long as the error in the DE is reduced everywhere in the domain during each training iteration.

1.2 Overview of the proposed method for solving BVPs

This work further develops an alternative method for numerical solution of BVPs already in the contemporary literature. Understanding the practical steps involved in implementing this method can get lost in the mathematical notation and text of the pages that follow. Figure 1 illustrates a flow chart outlining the process for initializing and training a BVP solution using ANNs. The process begins by identifying the BVP to be solved and ends with an approximate solution that is continuous over the entire problem domain.

The lightly shaded boxes in Figure 1 represent actions which must be taken by the user. The most significant of these involves modeling of the boundary and BCs. The boundary must be broken in a finite number of segments which may have any shape. Accurate solutions are generally easier to develop for domains with fewer boundary segments. Differentiable functions must then be defined which return the correct BCs and the normal vector for each segment. A length factor for each segment, representing a "distance" function from that segment, is then determined. A standard length factor definition for segments of arbitrary shape is presented in this work, but the user may choose any convenient length factor fulfilling three simple requirements.

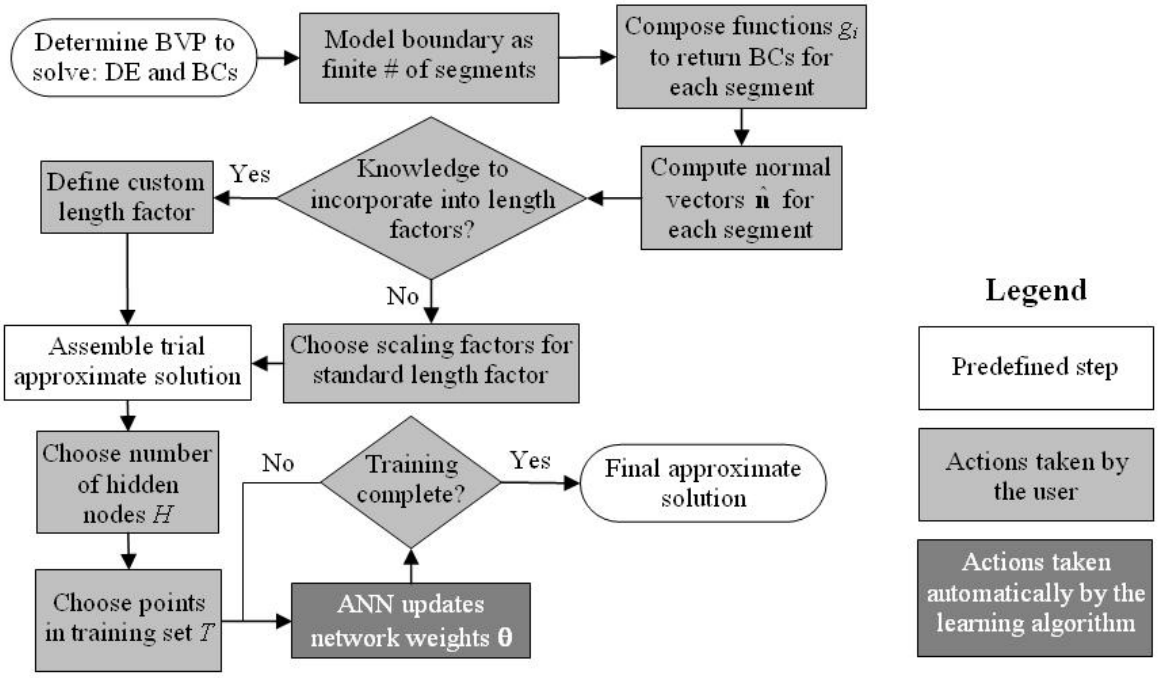


Figure 1. Flow chart outlining the process for developing an ANN solution to a BVP.

The user then chooses several design parameters. Scaling factor(s) for use in the standard length factor definition are chosen to minimize the effect of singularities, usually singularities in the length factor second partial derivatives at segment endpoints. Sufficiently many nodes in the hidden layer of the ANN must be chosen to provide sufficient computing power for the ANN to solve the BVP. Finally, a set of representative locations in the domain is identified to serve as the training set for optimizing the ANN.

Training of the ANN then commences where the learning algorithm automatically updates the network weights to produce a better trial approximate solution. Training follows an iterative process which can be thought of as a feedback loop. The training algorithm continually polls how well the DE is satisfied at the training locations and uses this information to update the ANN weights. The DE is again evaluated using the new TAS, and network weights are again modified. This process continues until training

produces no change in the TAS, or overfitting is observed. Overfitting is evident when satisfaction of the DE improves at the training locations, but worsens at other locations in the domain. Training with well-chosen design parameters is observed to produce an accurate final approximate solution.

1.3 Organization of this work

The first five chapters in this dissertation comprise work collected from other sources which is then applied to the current research. While content in these five chapters reflects significant effort by the author, it consists mainly of reapplication of existing techniques and does not represent the breaking of new ground. The basics of multi-layer perceptron networks and the relevant details for their application to solving BVPs are first presented. A case is then made for the benefits of developing a method which automatically satisfies the BCs specified by the BVP. Finally, the promise of weight reuse for accelerated training is explored.

Chapter 6 begins the core of the presented research; this innovative work represents a significant contribution to the fields of ANNs and numerical solution of BVPs. A major aspect of this work has been in successfully developing a TAS which automatically satisfies all BCs, Dirichlet and/or Neuman, for arbitrary irregular domain shapes. The intellectual process resulting in the final form of the TAS is documented. Definition of the length factors which enabled development of the TAS is then covered. Computational results follow for solving several BVPs, including consideration of non-homogenous and nonlinear DEs. Chapter 9 presents a theorem guaranteeing convergence toward the correct analytical solution if the error in the DE is reduced everywhere in the domain during all steps of training. A framework for software implementation of the

method for solving BVPs is then provided. Finally, the entire effort is concluded and recommendations made for future work.

2 Basics of multi-layer perceptron networks

2.1 Network structure

Multi-layer perceptron (MLP) networks are a type of artificial neural network (ANN) consisting of a number of interconnected nodes, or *perceptrons*. MLP feedforward neural networks are universal approximators [14] which are created by optimizing the network parameters, or weights, to minimize an error function. The output of a given node is a continuous function, called the transfer function, of the biased and weighted sum of all the inputs to the node. Two transfer functions commonly in use are the sigmoid function and the identity function. Sigmoid transfer functions constrain the nodal output between two values. An example is the logarithmic sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{3}$$

illustrated in Figure 2. The identity transfer function allows for unconstrained nodal output as a simple weighted sum of the inputs. The output of a node is then $f(z)$, either a sigmoid as in (3), or the identity function. In both cases, z is defined as

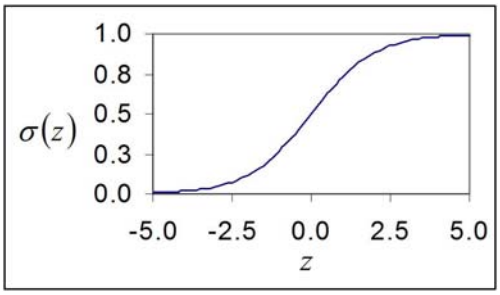


Figure 2. The logarithmic sigmoid function.

$$z = b + \sum_{i=1}^{\text{Node Inputs}} w_i x_i \tag{4}$$

where x_i is the i^{th} input to the node. The parameters for the node are the weights w_i and the bias b. All of the parameters, including biases, are generally called weights since a bias can be considered as a weight operating on a constant input.

Feedforward networks involve connections that always move towards the network output, i.e. there are no feedback loops. The nodes of a neural network are most often organized into layers where connections exist only between nodes in adjacent layers. A fully connected network contains links between all nodes in adjacent layers. A multilayer network contains at least one layer between the network input and output, i.e. inputs are not fed directly to output nodes. Figure 3 illustrates a fully connected two-input, single-output, feedforward, multilayer network with a single hidden layer consisting of three nodes.

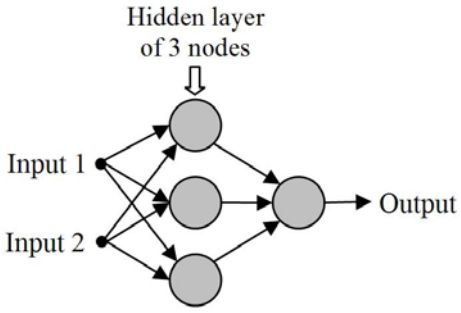


Figure 3. Schematic diagram of a fully connected feedforward MLP network with three hidden nodes.

2.2 Supervised learning algorithms

Supervised training of ANNs involves reduction of some error function, generally of the form

$$E = \frac{1}{|T|}\sum_{\mathbf{x}\in T} G(\mathbf{x}, N)^2 \tag{5}$$

where G is some signed error measure, T is a set of training points at which error is to be evaluated, and N is the output of the ANN. An optimization algorithm is then applied to adjust the ANN weights in order to minimize E. The most common method for optimizing network weights involves gradient descent. The sensitivity of E to each weight is computed for every input vector in T, and the weight values are adjusted so that overall error is decreased. Numerous so-called learning algorithms exist for determining weight change during an iteration. A good learning algorithm attempts to avoid the local minima of the highly multi-dimensional weight-space (the simple network in Figure 3 has thirteen weights!) while quickly converging to a solution with small error. Weight updating is repeated for a number of iterations, also called epochs, until the error is acceptably small or until a predetermined number of epochs have passed. The process of iteratively updating the network weights is called *training*.

The speed at which a network is trained depends heavily on the learning algorithm and the structure of the network. Generally, learning rules have design parameters that affect training speed. Additionally, network structure, including the number of nodes/layers as well as connections (nodes need not be fully connected for many problems), affect training time in two ways: more nodes and connections require more weights to optimize, and networks with too few nodes/connections may not have a sufficient number of degrees of freedom to quickly and/or accurately solve a complicated problem. So even though multilayer feedforward neural networks are universal approximators, a poorly designed neural network can still yield unsatisfactory results.

2.2.1 The RPROP learning algorithm

The RPROP algorithm [30] was originally chosen as the gradient descent method used in this effort due to its simplicity and the adaptive nature of its parameters. Consider a vector

$$\boldsymbol{\theta} = \left[\theta_1, \theta_2 \ldots, \theta_W\right] \tag{6}$$

containing the W network weights to be optimized.. Each weight θ_i has an update-value Δ_i associated with it, which is added to or subtracted from the current weight value depending on the sign of $\partial E/\partial \theta_i$. The weight in the t^{th} epoch is updated using

$$\theta_i^{(t)} = \begin{cases} \theta_i^{(t-1)} - \Delta_i^{(t-1)}, & \dfrac{\partial E}{\partial \theta_i^{(t)}} > 0 \\ \theta_i^{(t-1)} + \Delta_i^{(t-1)}, & \dfrac{\partial E}{\partial \theta_i^{(t)}} < 0 \end{cases} \tag{7}$$

The update-value is subtracted from the current weight if $\partial E/\partial \theta_i$ is positive, and added if negative. This approach is computationally inexpensive and quite powerful since the update-values are adjusted dynamically rather than depending on the magnitude of $\partial E/\partial \theta_i$ to determine the amount of weight update.

The update-value in the t^{th} epoch is adjusted according to

$$\Delta_i^{(t)} = \begin{cases} \eta^+ \Delta_i^{(t-1)}, & \dfrac{\partial E}{\partial \theta_i^{(t-1)}} \cdot \dfrac{\partial E}{\partial \theta_i^{(t)}} > 0 \\ \eta^- \Delta_i^{(t-1)}, & \dfrac{\partial E}{\partial \theta_i^{(t-1)}} \cdot \dfrac{\partial E}{\partial \theta_i^{(t)}} < 0 \end{cases} \tag{8}$$

The same sign of $\partial E/\partial \theta_i$ in the current and previous epochs indicates that adjustment should be accelerated, and thus the current update-value is increased by a factor of $\eta^+ > 1$. Convergence speed is not sensitive to small changes in η^+ [30] and the

recommended value of $\eta^+ = 1.2$ is used. A minimum has been passed in the previous epoch if the sign of $\partial E/\partial \theta_i$ changes in consecutive epochs. In this case, the update-value is reduced by a factor $0 < \eta^- < 1$ in order to converge on the minimum. Without any other knowledge of the minimum's location, the update in the previous epoch is reduced by half with a value $\eta^- = 0.5$.

Not only are the update-values adaptive, but they are specific for each weight. This enables fast convergence with limited computational cost, with the only drawback being the memory required to store the $\Delta_i^{(t-1)}$ and $\partial E/\partial \theta_i^{(t-1)}$ values for each weight. This memory requirement is not significant unless ANNs with thousands of nodes are considered. Another strength of this method is that convergence speed is not especially sensitive to the three parameter values $\eta^+ = 1.2$, $\eta^- = 0.5$, and $\Delta_0 = 0.1$ [30]. Figure 4 illustrates schematically update-value adaptation as the weight converges to its optimum value.

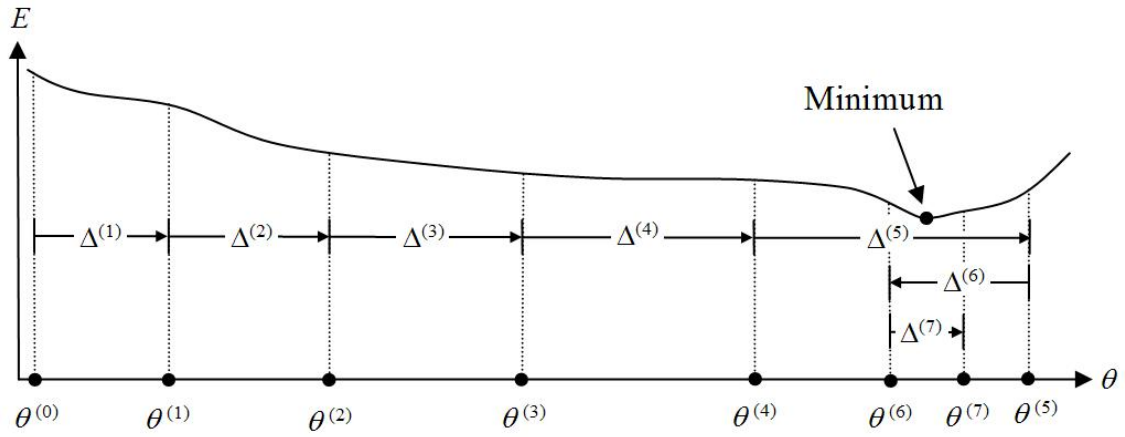


Figure 4. Schematic illustration of the first seven epochs of updating one weight using RPROP.

2.2.2 The Levenberg-Marquardt learning algorithm

Although conceptually simple and computationally inexpensive, the RPROP algorithm failed to sufficiently reduce error when solving more complicated BVPs. The Levenberg-Marquardt method is a classical approach known for its fast convergence when using a sum-of-error-squares error function as in (5). The Levenberg-Marquardt method [31] is derived by first considering the error E after a differential change in the ANN weights from $\boldsymbol{\theta}_0$ to $\boldsymbol{\theta}$ according to the second-order Taylor series

$$E(\boldsymbol{\theta}) = E(\boldsymbol{\theta}_0) + \mathbf{g}^{\mathrm{T}}(\boldsymbol{\theta} - \boldsymbol{\theta}_0) + \tfrac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^{\mathrm{T}} \mathbf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0) + \text{H.O.T.} \tag{9}$$

where

$$\mathbf{g} = \nabla E(\boldsymbol{\theta}) = \frac{\partial E}{\partial \boldsymbol{\theta}} = \left[\frac{\partial E}{\partial \theta_1}, \frac{\partial E}{\partial \theta_2} \cdots, \frac{\partial E}{\partial \theta_W} \right]^{\mathrm{T}} \tag{10}$$

is the gradient vector and

$$\mathbf{H}(\boldsymbol{\theta}) = \begin{bmatrix} \dfrac{\partial^2 E(\boldsymbol{\theta})}{\partial \theta_1^2} & \dfrac{\partial^2 E(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_2} & \cdots & \dfrac{\partial^2 E(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_W} \\ \dfrac{\partial^2 E(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_1} & \dfrac{\partial^2 E(\boldsymbol{\theta})}{\partial \theta_2^2} & \cdots & \dfrac{\partial^2 E(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_W} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 E(\boldsymbol{\theta})}{\partial \theta_W \partial \theta_1} & \dfrac{\partial^2 E(\boldsymbol{\theta})}{\partial \theta_W \partial \theta_2} & \cdots & \dfrac{\partial^2 E(\boldsymbol{\theta})}{\partial \theta_W^2} \end{bmatrix} \tag{11}$$

is the Hessian matrix.

The weight vector $\boldsymbol{\theta}^*$ corresponding to the minimum error is solved for by setting the gradient of (9) with respect to $\boldsymbol{\theta}$

$$\nabla E(\boldsymbol{\theta}^*) = \mathbf{H}(\boldsymbol{\theta}^* - \boldsymbol{\theta}_0) + \mathbf{g} \tag{12}$$

equal to zero and solving for

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathbf{H}^{-1}\mathbf{g} \tag{13}$$

which results in the Newton-Raphson learning algorithm

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta_i \mathbf{H}_i^{-1}\mathbf{g}_i \tag{14}$$

where $\boldsymbol{\theta}_i$ is the weight vector in the i^{th} iteration and η is the learning rate used to scale the magnitude of the weight change. This method will rapidly converge to the error minimum, assuming that the original $\boldsymbol{\theta}_0$ is already sufficiently close to $\boldsymbol{\theta}^*$.

The error in (9) is the same error as defined in (5), which is dependent on the choice of training points in T as well as the vector of ANN weights $\boldsymbol{\theta}$. The error $E(\boldsymbol{\theta})$ is considered only a function of $\boldsymbol{\theta}$ for purposes here since membership in T is not affected by the learning algorithm. The error in (5) can be rewritten as

$$E(\boldsymbol{\theta}) = \mathbf{G}(\boldsymbol{\theta})^{\mathrm{T}}\,\mathbf{G}(\boldsymbol{\theta}) \tag{15}$$

where $\mathbf{G}(\boldsymbol{\theta}) = \left[G(\mathbf{x}_1, \boldsymbol{\theta}), G(\mathbf{x}_2, \boldsymbol{\theta}) \ldots, G\left(\mathbf{x}_{|T|}, \boldsymbol{\theta}\right)\right]^{\mathrm{T}}$ given that $\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_{|T|} \in T$. Note that T is the matrix transpose operator and T is a set of points. Equation (15) indicates that error is computed by summing the square of G at each location in the domain as determined by T. This makes (5) and (15) equivalent except for the scalar scaling factor $|T|$. Inclusion of the scalar $|T|$ into the learning rate η negates any effect of its appearance in (5). The Jacobian matrix

$$\mathbf{J}(\mathbf{x},\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial G(\mathbf{x}_1,\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial G(\mathbf{x}_1,\boldsymbol{\theta})}{\partial \theta_2} & \cdots & \frac{\partial G(\mathbf{x}_1,\boldsymbol{\theta})}{\partial \theta_W} \\ \frac{\partial G(\mathbf{x}_2,\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial G(\mathbf{x}_2,\boldsymbol{\theta})}{\partial \theta_2} & \cdots & \frac{\partial G(\mathbf{x}_2,\boldsymbol{\theta})}{\partial \theta_W} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial G(\mathbf{x}_{\|T\|},\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial G(\mathbf{x}_{\|T\|},\boldsymbol{\theta})}{\partial \theta_2} & \cdots & \frac{\partial G(\mathbf{x}_{\|T\|},\boldsymbol{\theta})}{\partial \theta_W} \end{bmatrix} \tag{16}$$

is used to defined the Hessian

$$\mathbf{H} = 2\mathbf{J}^{\mathrm{T}}\mathbf{J} + 2\frac{\partial \mathbf{J}^{\mathrm{T}}}{\partial \boldsymbol{\theta}}\mathbf{G} \tag{17}$$

for the special case for sum-squared-error as in (15). The errors can be linearly approximated to produce $\mathbf{H} \approx 2\mathbf{J}^{\mathrm{T}}\mathbf{J}$, which constitutes a significant simplification for computation since it contains no mixed partial derivatives of $\boldsymbol{\theta}$. This approximation combined with (14) produces the Gauss-Newton learning algorithm

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta\mathbf{H}_i^{-1}\mathbf{g}_i = \boldsymbol{\theta}_i - \tfrac{1}{2}\eta_i\left(\mathbf{J}_i^{\mathrm{T}}\mathbf{J}_i\right)^{-1}\mathbf{g}_i \tag{18}$$

The Gauss-Newton method assumes that the second term on the right-hand side of (17) is negligible, a fact which is not always true. The Levenberg-Marquardt learning algorithm modifies (18) to

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta\mathbf{H}_i^{-1}\mathbf{g}_i = \boldsymbol{\theta}_i - \tfrac{1}{2}\eta_i\left(\mathbf{J}_i^{\mathrm{T}}\mathbf{J}_i + \lambda_i\mathbf{I}\right)^{-1}\mathbf{g}_i \tag{19}$$

where λ is a scalar and $\mathbf{I}$ is the (T,T) identity matrix. This learning algorithm reduces to Gauss-Newton as λ approaches zero, and reduces to steepest decent in the limit

$$\lim_{\lambda_i \to \infty} \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \tfrac{1}{2}\eta_i\left(\frac{1}{\lambda_i}\right)\mathbf{g}_i = \boldsymbol{\theta}_i - \frac{\eta_i}{2\lambda_i}\mathbf{g}_i \tag{20}$$

The result is a learning algorithm with a direction for $d\theta$ somewhere between the steepest descent and Gauss-Newton directions as specified by λ.

The learning rate η and factor λ are designed to adapt dynamically during training. The strategy is usually to choose large values for these parameters at the beginning of training and to decrease them as the solution improves. In such a way, large learning rates allow ANN weights to quickly change in the beginning of training and then fine tune themselves towards the end. Training commences with higher λ values, effectively implementing steepest descent until the solution becomes closer to the minimum error solution when the Gauss-Newton method is more effective. Computations in this work have been kept as simple as possible, using static values of $\eta = \lambda = 0.01$ for these parameters.

The Levenberg-Marquardt learning algorithm is a classical gradient descent method exploiting the sum-of-error-squares function to provide a fast-converging and effective optimization scheme for ANN weights. Although not as simple and intuitive as RPROP, this proven method is more powerful and still simple to implement using a few basic matrix operations. Unless otherwise specified, this learning algorithm was used to generate all computational results reported here.

3 The error function and its gradient for solving BVPs

3.1 Error function definition

The parameter weights in the ANN are iteratively updated using gradient decent in order to produce a function as close as possible to the analytical solution of the DE considered. The resulting TAS is developed by minimizing an error function using the gradient of error with respect to each of the ANN weights. The TAS $\psi_t(\mathbf{x}, N)$ is a function of the desired location **x** in the domain, and the neural network output $N(\mathbf{x}, \boldsymbol{\theta})$. This TAS is developed by minimizing the error function

$$E = E_{\mathrm{DE}} + \eta E_{\mathrm{BC}} \tag{21}$$

where the first term

$$E_{\mathrm{DE}} = \frac{1}{|S|} \sum_{\mathbf{x} \in S} \left[\tilde{D}\psi_t(\mathbf{x}) - f(\mathbf{x}) \right]^2 \tag{22}$$

accounts for error in approximating the DE itself, and in the second term

$$E_{\mathrm{BC}} = \frac{1}{|S_D|} \sum_{\mathbf{x} \in S_D} \left[\psi_t(\mathbf{x}) - g_D(\mathbf{x}) \right]^2 + \frac{1}{|S_M|} \sum_{\mathbf{x} \in S_M} \left[\hat{\mathbf{n}}(\mathbf{x}) \cdot \nabla \psi_t(\mathbf{x}) - g_M(\mathbf{x}) \right]^2 \tag{23}$$

accounts for error in satisfying the Dirichlet (*D*) and Neuman (*M*) BCs. The weighting factor η determines the relative importance of the two error components. Equations (22) and (23) are defined where *S* is composed of a finite set of points within the domain, S_D is the set of points where the boundary value $g_D(\mathbf{x})$ is specified, S_M is the set of points where the boundary gradient $g_M(\mathbf{x})$ is specified, and $\hat{\mathbf{n}}(\mathbf{x})$ is the inwardly directed unit normal to the boundary at **x**. Error defined in (21) follows the same convention used in

[20, 32-34] with the added possibility of Neuman boundary conditions introduced by the second term in the right-hand side of (23).

3.2 Development of the error gradient

The TAS $\psi_t(\mathbf{x}, N)$ is a known function of the input vector $\mathbf{x}$ and the ANN output $N(\mathbf{x}, \boldsymbol{\theta})$. Error in (21) is minimized by adjusting the weights composing $\boldsymbol{\theta}$. The gradient $\partial E / \partial \boldsymbol{\theta}$ must first be calculated in order to employ a gradient descent approach for iteratively updating the ANN weights. The gradients of the two error components are

$$\frac{\partial E_{\mathrm{DE}}}{\partial \boldsymbol{\theta}} = \frac{2}{|S|} \sum_{\mathbf{x} \in S} \left[\tilde{D}\psi_t(\mathbf{x}) - f(\mathbf{x}) \right] \frac{\partial \left[\tilde{D}\psi_t(\mathbf{x}) - f(\mathbf{x}) \right]}{\partial \boldsymbol{\theta}} \Bigg|_{\mathbf{x}} \quad \text{and} \tag{24}$$

$$\begin{aligned} \frac{\partial E_{\mathrm{BC}}}{\partial \boldsymbol{\theta}} = & \frac{2}{|S_D|} \sum_{\mathbf{x} \in S_D} \left[\psi_t(\mathbf{x}) - g_D(\mathbf{x}) \right] \frac{\partial \psi_t}{\partial \boldsymbol{\theta}} \Bigg|_{\mathbf{x}} + \\ & \frac{2}{|S_M|} \sum_{\mathbf{x} \in S_M} \left[\hat{\mathbf{n}}(\mathbf{x}) \cdot \nabla \psi_t(\mathbf{x}) - g_M(\mathbf{x}) \right] \frac{\partial \left(\hat{\mathbf{n}} \cdot \nabla \psi_t \right)}{\partial \boldsymbol{\theta}} \Bigg|_{\mathbf{x}} \end{aligned} \tag{25}$$

Each term appearing in (24) and (25) will be some function of $\mathbf{x}$ multiplied by a certain combination of partial derivatives of ψ_t as given by

$$f(\mathbf{x}) \frac{\partial^{\sum_{j=1}^{J} \lambda_j} \psi_t}{\partial x_1^{\lambda_1} \partial x_2^{\lambda_2} \cdots \partial x_J^{\lambda_J}} \tag{26}$$

where λ_j is the order of the partial derivative with respect to x_j. The gradient of each of these terms with respect to the ANN weights then takes the form

$$f(\mathbf{x}) \frac{\partial^{1+\sum_{j=1}^{J} \lambda_j} \psi_t}{\partial x_1^{\lambda_1} \partial x_2^{\lambda_2} \cdots \partial x_J^{\lambda_J} \partial \boldsymbol{\theta}} \tag{27}$$

The terms in (24) and (25) are special cases of (27) and thus the gradient $\partial E/\partial \boldsymbol{\theta}$ will consist of numerous terms with mixed partial derivatives of ψ_t with respect to x_j and $\boldsymbol{\theta}$. The method for determining these mixed partials begins by successively performing the necessary derivatives with respect to any given x_j. In order to perform the partial derivatives of ψ_t with respect to x_j, consider the more general case where ψ_t or any of its derivatives are functions of $U(\mathbf{x})$ and $N(\mathbf{x},\boldsymbol{\theta})$ rather than simply $\mathbf{x}$ and $N(\mathbf{x},\boldsymbol{\theta})$. The partial derivative with respect to x_j is then

$$\frac{\partial}{\partial x_j} = \frac{\partial U}{\partial x_j}\frac{\partial}{\partial U} + \frac{\partial N}{\partial x_j}\frac{\partial}{\partial N} \tag{28}$$

which reduces to

$$\frac{\partial}{\partial x_j} = \left.\frac{\partial}{\partial x_j}\right|_{N=\text{const}} + \frac{\partial}{\partial N}\frac{\partial N}{\partial x_j} \tag{29}$$

when $U = x_j$ as is the case here. Any terms in N or its partial derivatives appearing in the result are then differentiated with respect to $\boldsymbol{\theta}$ in order to complete the required mixed partial derivative. This involves mixed partial derivatives of N in x_j and $\boldsymbol{\theta}$. These partial derivatives are dependent on the architecture of the ANN in question, which is the topic of the next section.

3.3 Partial derivatives of the ANN output

The ANN architecture from [21, 27] is adopted here, which consists of a single layer of H biased hidden nodes with logarithmic sigmoid transfer functions connected to a single unbiased identity output node, as illustrated in Figure 5. The output of this ANN is

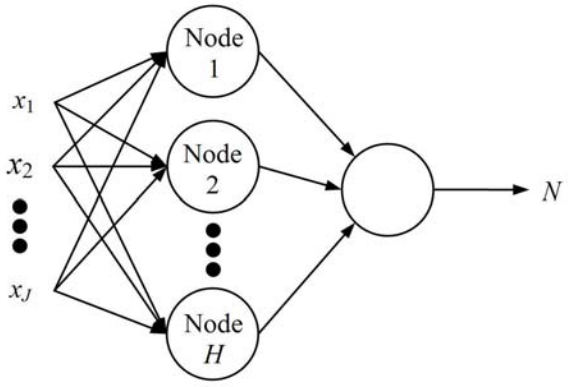


Figure 5. Single-output architecture of the ANN used.

$$N = \sum_{i=1}^{H} v_i \sigma\left(z_i\right) \tag{30}$$

given that

$$z_i = u_i + \sum_{j=1}^{J} w_{ij} x_j \tag{31}$$

where w_{ij} is the weight connecting the j^{th} input to the i^{th} hidden node, u_i is the bias of the i^{th} hidden node, and v_i is weight connecting the i^{th} hidden node to the output node. The vector

$$\boldsymbol{\theta} = \left[u_1, u_2 \ldots, u_H, v_1, v_2 \ldots, v_H, w_{11}, w_{12} \ldots, w_{1I}, w_{21}, w_{21} \ldots, w_{2J} \ldots, w_{H1}, w_{H2} \ldots w_{HJ}\right]^{\mathrm{T}} \tag{32}$$

contains all of the parameter weights which must be optimized to produce the TAS.

Mixed partial derivatives of the forms

$$\frac{\partial^{1+\sum_{j=1}^{J}\lambda_j} N}{\partial x_1^{\lambda_1} \partial x_2^{\lambda_2} \cdots \partial x_J^{\lambda_J} \partial u_i}, \tag{33}$$

$$\frac{\partial^{1+\sum_{j=1}^{J}\lambda_j} N}{\partial x_1^{\lambda_1} \partial x_2^{\lambda_2} \cdots \partial x_J^{\lambda_J} \partial v_i}, \text{ and} \tag{34}$$

$$\frac{\partial^{1+\sum_{j=1}^{J}\lambda_j} N}{\partial x_1^{\lambda_1}\partial x_2^{\lambda_2}\cdots\partial x_J^{\lambda_J}\partial w_{ij}} \tag{35}$$

are required to complete the gradient $\partial E/\partial \boldsymbol{\theta}$ developed in the previous section. These partial derivatives are obtained using the chain rule, which requires computation of the partial derivative

$$\frac{\partial z_i}{\partial x_j} = \frac{\partial}{\partial x_j}\left(u_i + \sum_{k=1}^{J} w_{ik}x_k\right) = w_{ij} \tag{36}$$

and the various derivatives of the logarithmic sigmoid function σ. The first three logarithmic sigmoid derivatives are

$$\sigma'(z) = -\left(1+e^{-z}\right)^{-2}\left(-e^{-z}\right) = e^{-z}\sigma^2, \tag{37}$$

$$\sigma''(z) = -e^{-z}\sigma^2 + 2e^{-z}\sigma\sigma' = 2e^{-2z}\sigma^3 - e^{-z}\sigma^2, \text{ and} \tag{38}$$

$$\sigma'''(z) = -4e^{-2z}\sigma^3 + 6e^{-2z}\sigma^2\sigma' + e^{-z}\sigma^2 - 2e^{-z}\sigma\sigma' = 6e^{-3z}\sigma^4 - 6e^{-2z}\sigma^3 + e^{-z}\sigma^2 \tag{39}$$

The partial derivative of the ANN output with respect to input x_j can then be written as

$$\frac{\partial N}{\partial x_j} = \sum_{k=1}^{H} v_k \frac{\partial}{\partial x_j}\sigma(z_k) = \sum_{k=1}^{H} v_k\sigma'(z_k)\frac{\partial z_k}{\partial x_j} = \sum_{k=1}^{H} v_k w_{kj}\sigma'(z_k) \tag{40}$$

which is extended to the general case

$$\frac{\partial^{\sum_{j=1}^{J}\lambda_j} N}{\partial x_1^{\lambda_1}\partial x_2^{\lambda_2}\cdots\partial x_J^{\lambda_J}} = \sum_{k=1}^{H}\left(v_k\sigma^{(\Lambda)}(z_k)\prod_{j}^{J} w_{kj}^{\lambda_j}\right) = \sum_{k=1}^{H} v_k P_k \sigma_k^{(\Lambda)} \tag{41}$$

with the definitions

$$\Lambda \equiv \sum_{j=1}^{J}\lambda_j,\ P_i \equiv \prod_{j}^{J} w_{ij}^{\lambda_j},\ \text{and}\ \sigma_i^{(\Lambda)} \equiv \sigma^{(\Lambda)}(z_i) \tag{42}$$

Differentiating (41) with respect to v_i eliminates all but the i^{th} term to generate

$$\frac{\partial^{1+\sum_{j=1}^{J}\lambda_j} N}{\partial x_1^{\lambda_1}\partial x_2^{\lambda_2}\cdots\partial x_J^{\lambda_J}\partial v_i} = P_i\sigma_i^{(\Lambda)} \tag{43}$$

Only the $\sigma_i^{(\Lambda)}$ term in (41) depends on u_i, and differentiating with respect to u_i produces

$$\frac{\partial^{1+\sum_{j=1}^{J}\lambda_j} N}{\partial x_1^{\lambda_1}\partial x_2^{\lambda_2}\cdots\partial x_J^{\lambda_J}\partial u_i} = v_i P_i\sigma_i^{(\Lambda+1)} \tag{44}$$

since $\partial z_i/\partial u_i = 1$. The derivative with respect to the hidden weight w_{ij} is more complicated since both P_i and $\sigma_i^{(\Lambda)}$ are functions of it. These derivatives are

$$\begin{aligned}
\frac{\partial^{1+\sum_{k=1}^{J}\lambda_k} N}{\partial x_1^{\lambda_1}\partial x_2^{\lambda_2}\cdots\partial x_J^{\lambda_J}\partial w_{ij}} &= \sum_{k=1}^{H} v_k \frac{\partial}{\partial w_{ij}}\left(P_k\sigma_k^{(\Lambda)}\right) = v_k \frac{\partial}{\partial w_{ij}}\left(P_i\sigma_i^{(\Lambda)}\right) \\
&= v_i P_i \frac{\partial\sigma_i^{(\Lambda)}}{\partial w_{ij}} + v_i\sigma_i^{(\Lambda)}\frac{\partial P_i}{\partial w_{ij}} \\
&= v_i P_i\sigma_i^{(\Lambda+1)}\frac{\partial z_i}{\partial w_{ij}} + v_i\sigma_i^{(\Lambda)}\frac{\partial}{\partial w_{ij}}\left(\prod_k^J w_{ik}^{\lambda_k}\right) \\
&= v_i x_j P_i\sigma_i^{(\Lambda+1)} + v_i\lambda_j w_{ij}^{\lambda_j-1}\sigma_i^{(\Lambda)}\prod_{k=1,k\neq j}^{J} w_{ik}^{\lambda_k}
\end{aligned} \tag{45}$$

since $\partial z_i/\partial w_{ij} = x_j$.

All the partial derivatives of N required for the calculation of $\partial E/\partial\boldsymbol{\theta}$ are now defined, enabling the use of gradient descent for updating the network weights. The partial derivatives of N do not depend on the DE and are thus valid as long as the neural network architecture is not modified aside from changing H, the number of hidden nodes. Notice that the highest order derivative of the sigmoid function is $\Lambda+1$ and thus

the first three derivatives appearing in (37) through (39) are sufficient for any DE of second order or less.

4 Improvement through automatic satisfaction of BCs

The idea of improving performance of iterative numerical BVP solution techniques by automatically satisfying BCs is not new. One method [35] uses blended bivariate interpolation [36] in order to define an approximate solution exactly matching BCs. This approximation is then used to initialize iterative methods such as the Gauss-Seidel method [37] for solving BVPs using a finite difference approach. Matching BCs before beginning the iterative process was shown to reduce the number of iterations necessary to produce an acceptable approximation of the analytical solution. Reference [35] also incorporates other known characteristics of the correct solution, such as convexity or satisfaction of a maximum principle for example, into the approximate solution to improve performance of the iterative algorithm. These are also goals of the research reflected in this work, only using an ANN method for solving BVPs.

Exact matching of BCs with ANN methods has also been studied [27]. This method also uses blended bivariate interpolation to satisfy BCs and then adds another term involving ANN output in order to adjust the values of the approximate solution on the interior of the domain. The remainder of this chapter illustrates how the performance of ANN methods can be enhanced by developing an approximate solution which automatically satisfies all BCs.

4.1 Comparison of two forms for the TAS

The previous chapter is reticent on the form of the TAS $\psi_t(\mathbf{x}, N)$, but assumes that the partial derivatives $\partial\psi_t/\partial x_j\big|_{N=\text{const}}$ and $\partial\psi_t/\partial N$ are available. The simplest and most common method is to use the ANN output directly as the TAS as in

$$\psi_t(\mathbf{x}, N) = N. \tag{46}$$

However, the TAS can be judiciously redefined [27] to take the form

$$\psi_t(\mathbf{x}, N) = A(\mathbf{x}) + F(\mathbf{x}, N) \tag{47}$$

where $A(\mathbf{x})$ is chosen to satisfy the BCs exactly and $F(\mathbf{x}, N)$ is chosen to be zero for any $\mathbf{x}$ on the boundary. This produces a TAS which automatically satisfies the BCs regardless of the ANN output. This method forces E_{BC} to zero, thus eliminating the second term in the error function from (21). A systematic approach for determining the functions A and F exists for arbitrary BCs on a rectangular domain [27].

A comparison [28] by the present author of DEs solved with the TASs in (46) and (47) investigated whether eliminating BCs with (47) improves TAS accuracy for the ordinary differential equation initial-value problem

$$2\frac{d^2\psi}{dx^2} + \frac{d\psi}{dx} + 2x = 0 \text{ where } \psi(0) = 1 \text{ and } \left.\frac{d\psi}{dx}\right|_{x=0} = 0 \tag{48}$$

on the domain [0,1], as well as the diffusion and Laplace partial differential equation BVPs

$$\frac{1}{4}\frac{\partial^2\psi}{\partial x_1^2} - \frac{\partial\psi}{\partial x_2} = 0 \text{ where } \psi(0, x_2) = \psi(1, x_2) = 0 \text{ and } \psi(x_1, 0) = \sin\pi x_1 \tag{49}$$

$$\frac{\partial^2\psi}{\partial x_1^2} + \frac{\partial^2\psi}{\partial x_2^2} = 0 \text{ where } \psi(x_1, 0) = \psi(0, x_2) = \psi(1, x_2) = 0 \text{ and } \psi(x_1, 1) = \sin\pi x_1 \tag{50}$$

solved on unit square domains. The three BVPs in (48) through (50) have TASs

$$\psi_t = 1 + \psi^2 N \,, \tag{51}$$

$$\psi_t = \sin \pi x_1 + x_1 x_2 \left(1 - x_1\right) N \,, \text{ and} \tag{52}$$

$$\psi_t = \left(1 - x_1\right) \sin \pi x_1 + x_1 x_2 \left(1 - x_1\right)\left(1 - x_2\right) N \tag{53}$$

respectively, when cast in the form of (47).

Figure 6 displays progression during training of the RMS difference between ψ_t and ψ when solving (48) through (50). All results were obtained by averaging performance over 25 runs generated by different random starting weights, and using 10 hidden nodes in each case with the RPROP learning algorithm. Ten evenly spaced points were chosen for S when solving (48), and 100 points on an evenly spaced 10×10 grid were chosen when solving (49) and (50). The BCs were evaluated at $S_D = S_M = \{x = 0\}$ when solving (48), and 10 evenly spaced points along each appropriate boundary axis were chosen for S_D when solving (49) and (50). The data with shaded markers represent solving the BVPs with the simple TAS in (46) and the unshaded markers with (47).

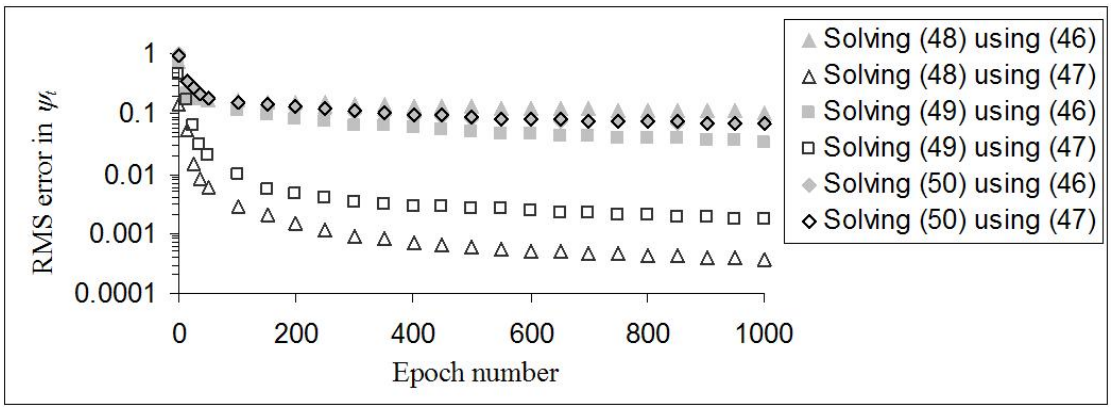


Figure 6. Comparison of error for three different BVPs for the TAS forms in (46) and (47).

Figure 6 illustrates that automatically satisfying only the initial position and

velocity in (48) does not significantly aid the ANN in finding optimal parameters since the shaded and unshaded diamond markers nearly coincide throughout training. However, exact BCs enforced along entire axes ease determination of optimal ANN parameters for (49) and (50). Improvement was less pronounced for (49) than for (50) since the former has three BCs whereas the latter has four. These results suggest then that determination of optimal ANN parameters is best accomplished when values around the entire solution domain boundary are automatically satisfied through the use of (47).

4.2 Comparison with RBF network methods from the literature

The TAS in (47) provides accurate results with a simpler implementation than other ANN methods, such as radial basis function (RBF) networks for solving DEs [22, 23]. RBF networks utilize nodes with local influence on the problem domain and require optimizing location and range of effect for each node in addition to the weights composing the θ parameter vector. In addition, [22] requires optimization of an additional design parameter with no guidance other than trial-and-error. Reference [23] requires no extra design parameter but introduces instead the additional complication of dynamically adding nodes during training. Both [22] and [23] solve the BVP

$$\frac{\partial^2 \psi}{\partial x_1^2} + \frac{\partial^2 \psi}{\partial x_2^2} = \sin\left(\pi x_1\right)\sin\left(\pi y_1\right) \tag{54}$$

subject to homogenous BCs $\psi\left(x_1, 0\right) = \psi\left(x_1, 1\right) = \psi\left(0, x_2\right) = \psi\left(1, x_2\right) = 0$. The analytical solution is then

$$\psi = -\frac{1}{2\pi}\sin\left(\pi x_1\right)\sin\left(\pi x_2\right) \tag{55}$$

The TAS of the form in (47) corresponding to these BCs,

$$\psi_t = x_1 x_2 \left(x_1 - 1\right)\left(x_2 - 1\right) N\left(\mathbf{x}, \boldsymbol{\theta}\right) \tag{56}$$

was used to train an ANN solution using a 5×5 evenly spaced grid for S, five evenly spaced points on each axis for S_D, and several numbers of hidden nodes with the RPROP algorithm. Fifty runs for each number of nodes were performed and the results averaged since the final parameter values are somewhat dependent on their random starting values. The performance of the solution is quantified by the error norm

$$E_{\text{norm}} = \sqrt{\frac{\sum_{\mathbf{x}\in T}\left[\psi\left(\mathbf{x}\right) - \psi_t\left(\mathbf{x}\right)\right]^2}{\sum_{\mathbf{x}\in T}\psi\left(\mathbf{x}\right)^2}} \tag{57}$$

where T is the set of points on a 100×100 grid. Figure 7 compares average performance over the fifty runs to results from [22] and [23], obtained using an ANN trained using RPROP with a 5×5 grid and consisting of 25 and 21 nodes, respectively. The upper error bar in Figure 7 represents one standard deviation while the lower bar indicates the minimum error of the fifty runs. Performance is essentially independent of the number of hidden nodes as long as sufficiently many are chosen. Not only does the simpler method outperform [22] and [23] with fewer nodes, but it enjoys the additional benefit that the BCs are exactly rather than approximately satisfied.

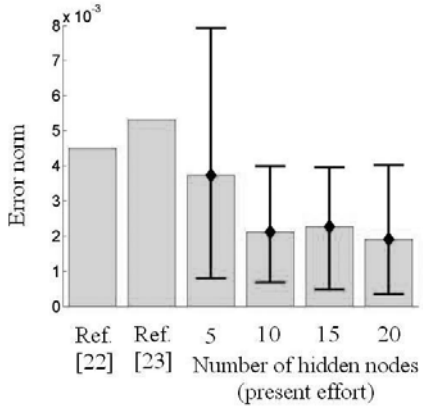


Figure 7. Error norm (57) with different numbers of hidden nodes when solving (55) compared with results from [22] and [23].

Reference [22] reduces its error norm further by a factor of just over ten by approximating the derivatives of ψ_t rather than ψ_t itself and integrating to generate the solution. This lower error is not significantly smaller than the lowest error from the fifty runs with 20 nodes. The mean error on the order of 10^{-3} should be sufficient for most engineering applications, but multiple runs can be performed to generate lower-error solutions if necessary.

4.3 Summary of the chosen form for the TAS

The remainder of the results presented in this work will use the form of the TAS in (47) where BCs are automatically satisfied before training commences. The results from this chapter justify use of (47) with its superior performance when compared both with (46) and with RBF networks from [22, 23]. The method for solving BVPs with (47) is simpler than other ANN approaches and yet generally produces better results since the TAS by definition is exact along Dirichlet boundaries, and correspondingly closer near those boundaries.

The E_{BC} term in (21) is necessarily zero for TASs of the form (47) where BCs are automatically satisfied; in such a case, (21) reduces to (5) and leads to defining G as

$$G \equiv \tilde{D}\psi_t\left(\mathbf{x}, N\right) - f\left(\mathbf{x}\right) \tag{58}$$

given a trial approximate solution ψ_t of the form in (47) and the set of training points $T = S$. The error in (5) will be used for the remainder of this effort for training the ANN. Note that the error used for training the ANN in (5) and the error norm for evaluating the fitness of the TAS in (57) both use a set of points in the domain named *T*. The set *T* used for each equation is usually different, with the set of points in (57) generally more extensive in order to judge the generalization/interpolation ability of the TAS. Remember that the error norm in (57) will not be available during training since the exact analytic solution is not known. Results from (57) are presented solely to judge the fitness of the TAS when developing the method, and would not be used in practice.

Since (57) is not available to evaluate the TAS during training, another metric must be employed for judging when to halt training. Otherwise, overfitting to the training data could occur where error in (5) reduces in each epoch, and yet the TAS worsens as the ANN overcompensates to reduce error at the training locations while ignoring regions in the domain between training data. Overfitting can be avoided by evaluating (5) during training, but with a set *T* including points different from, and more densely spaced than those used during training. This technique, discovered empirically by the present author, led to elaboration of the theorem presented later which guarantees convergence of the TAS toward the true solution if error in the DE is reduced everywhere in the domain.

5 Weight reuse

The original proposal of this effort emphasized weight reuse as major aspect of the research. In fact, solving BVPs was the application chosen specifically to explore the promise of weight reuse. As research progressed, it became apparent that solving BVPs with ANNs was not an exhausted field and many interesting facets remained to be explored, independent of the weight reuse issue. No ANN method has been found in the literature which can solve BVPs on arbitrary irregular domains with a TAS automatically satisfying all BCs. The main focus of research shifted to developing such a TAS, and weight reuse was left as an interesting topic for future research. This chapter includes the completed work in weight reuse, despite the change in focus, since weight reuse was important during the early stages of this research and even produced a contribution to the technical literature [28].

5.1 Motivation for weight reuse

Making the operation of ANNs more transparent is an active field of research which most notably includes hybridization with fuzzy logic [38]. The idea is that the linguistic power of fuzzy logic can infuse the otherwise black-box ANN with human-like knowledge. In such a way, ANNs are made to more closely resemble the thinking of human intelligence. Neuro-fuzzy hybrids are easily applicable to problems which involve a natural language rule base, such as in controls applications [39]. The present effort investigates using ANNs to solve BVPs [20-23, 27, 28, 32-34, 40-43], which unfortunately lack a linguistic component. A different approach is thus required to transfer human knowledge into the problem. One method for infusing knowledge is

through automatically satisfying the BCs of the BVP. This is accomplished for irregular domains with an innovative length factor introduced later in this dissertation. The resulting TAS simplifies the work required of the ANN by ensuring that the boundary conditions (BCs) are exactly satisfied everywhere along arbitrary irregular boundaries. It will be shown that the length factor not only helps to satisfy the BCs, but provides the ANN with a first-order approximation of the solution even inside the domain. Another method to impart knowledge about the BVP to the ANN is through weight reuse. With weight reuse, the ANN can make use of a previous solution to a similar BVP and apply it to a new, unknown problem.

An obvious component of human intelligence is the ability to learn through observation. A child lacks the skills to care for itself, and survives independently only after gaining sufficient experience to deal with the multitude of situations presented by life. And even though every new situation is unique, past experiences with similar conditions enable determination of an appropriate response. Soft computing techniques such as neural, fuzzy, and evolutionary computing generally undergo a period of *training* or *learning* after which the finished system has some ability to *generalize* by producing an appropriate response to stimuli not included during training. An example of such generalization is when the solution to a BVP is developed by reducing the error in the desired DE for a relatively small number of points within the domain, with expectation that the result will generalize to all regions within the domain.

This type of generalization is termed first-order generalization since the ANN learns to solve the same problem, but at locations in the domain for which it has no direct experience. Second-order generalization is equally important in the human

problem solving process; that is, applying knowledge of the solution of one problem when presented with a different and unknown problem. The technique of weight reuse in ANNs offers a method for second-order generalization when solving BVPs by reducing training time and decreasing error in the TAS when the problems considered are in some sense similar. Use of length factors and weight reuse together produce an ANN method for numerical solution of BVPs which more closely resembles the process of human thinking and learning.

Humans make good use of previously acquired knowledge when presented with a new challenge: parallels are drawn between similar problems already solved in order to attack a new one. One method for applying this observation in humans to training an ANN is through choice of the initial values for the network weights. The traditional approach is to initialize the weights as small random values. More sophisticated weight initialization techniques [44] can significantly increase the speed at which the ANN is trained to achieve an acceptable solution. These techniques are generally based on initializing weights to values that maximize the sensitivity of network output on the weights. The sensitivity is determined given a priori knowledge about the input/output relationship for the single problem at hand. However, weight initialization can be based on a previous ANN solution when solving a second, related problem [45]. In terms of BVPs, the basis for similarity could involve different BCs or a change in the equation coefficient(s). Simply employing the weights from the previously solved problem as initial weights for the related problem greatly accelerates network training when slightly changing either an equation coefficient or boundary condition [34].

5.2 Examples of weight reuse when solving BVPs

The BVPs solved in [34] consider only slight changes in a single boundary value or DE parameter. While improvements from weight reuse are not surprising for slight changes, the obvious question is how similar BVPs must be in order to benefit from weight reuse. Investigation into this question was begun by the present author [28] by comparing weight reuse for the TASs in both (46) and (47) when solving the DEs in (48) through (50) on rectangular domains. Results from [28] train the ANNs using RPROP as the learning algorithm.

Results with weight reuse for solving the DEs in (48) and (50) agreed with intuitive expectation for both types of TAS: weight reuse for DEs that were significantly different, but yet somewhat similar tended to produce accelerated training. For example, training was accelerated in the spring-mass-damper system in (46) when solving an under-damped problem after first learning another under-damped problem with different coefficients and the same initial conditions. However, starting with an over- or critically-damped problem produced no significant improvement when reusing weights to solve an under-damped problem.

While solving (48) and (50) produced intuitively predictable results, weight reuse when solving (49) departed from expectation. Weight reuse accelerated training as expected with the simple TAS in (46), but not with that in (47). In fact, weight reuse resulted in solutions significantly worse than starting with random weights, even for very similar problems! This unexpected failure of weight reuse using the TAS in (47) is surprising, especially since it successfully accelerated training when solving (48) and (50). Whether this failure is an isolated incident for one particular BVP is unknown;

numerical experiments with more varied DEs and BCs should shed light on the source of the failure and provide useful insight into weight reuse in general.

Weight reuse as described above utilizes the weights from a single previous ANN solution to solve a new, related problem. In fact, a single network can be developed which can learn a large number of DEs with different BCs [32]. Evolutionary algorithms are used to determine the weights for a network that solves all the problems within a class of DEs. However, only DEs with a different BC at a single point were considered in [32]. The present author's numerical experiments [28] show that classes of differential equations with respect to weight reuse are not so restricted, although exactly where class boundaries lie has not yet been established in the literature.

Weight reuse with the irregular domain TAS developed in the next chapter has not yet been tested, and is left for future research. The introduction of irregular domains adds another facet to weight reuse: the ANN could be retrained using a different but similar boundary shape as opposed to changing the DE and/or BCs. Investigation into improvements in solution fitness using weight reuse offer an interesting continuation of the research presented in this work.

6 Evolution of a viable TAS on irregular domains

Solving BVPs with a trial TAS in the form of (47) as introduced by [27] is established as a simple and effective method for rectangular domains. However, any ANN technique for solving BVPs must address irregular domains in order to compete with existing numerical techniques. Application to irregular domains introduces no further conceptual complications in the present environment since no restriction is placed on how the set of training points T is chosen.

ANN methods for solving BVPs on irregular domains could benefit from automatic satisfaction of BCs, just as do rectangular domains. However, correctly defining the functions $A(\mathbf{x})$ and $F(\mathbf{x},N)$ in (47) is problematic when considering arbitrary boundary shapes. The authors of [27] addressed this issue [21] but abandoned some of the simplicity and elegance of the original idea. Their method consists of a two-step process where the problem is first solved with the TAS in (46) and then a certain number of RBF nodes are added to adjust the solution in order to exactly satisfy the BCs at a discrete number of boundary points. The RBF centers are placed at points on the boundary where the BCs are to be satisfied, and the remaining RBF parameters are determined by solving a linear system. This method produces very low errors, but at the expense of an extra step which moves in the direction of finite element methods: BCs are satisfied exactly only at certain predefined locations (and approximated elsewhere) by solving linear systems with the associated issue of singularity.

Exactly matching BCs on arbitrary domain shapes was successfully implemented in [35] by mapping the irregular domain onto a rectangular domain [46, 47] before

applying the technique for rectangular domains. This approach succeeds when solving with the finite difference method in [35] since no need exists to insert the influence of the ANN output into the approximate solution. A "switch" function would be necessary to remove influence of the ANN output at locations on the domain boundary. The main challenge of the method proposed in this work arises from singularities at boundary segment intersections. This problem would appear in appropriate "switch" functions even when using domain mapping as in [35]. As a result, the added complication of domain mapping was not considered when extending the ANN technique in [27] to irregular domain shapes.

A major thrust of this research has been to define the functions $A(\mathbf{x})$ and $F(\mathbf{x},N)$ from (47) so that the TAS exactly satisfies BCs everywhere along boundaries of arbitrary shape for problems with mixed Dirichlet and Neuman BCs. Training ANN solutions to BVPs with such a TAS has since been shown to be simple and accurate, solving with a single gradient-descent optimization step. The idea was to break the function $A(\mathbf{x})$, which satisfies all BCs, into two parts

$$A(\mathbf{x}) = A_D(\mathbf{x}) + A_M(\mathbf{x}) \tag{59}$$

where $A_D(\mathbf{x})$ satisfies any Dirichlet BCs, and $A_M(\mathbf{x})$ satisfies any Neuman BCs. In addition to satisfying their appropriate BCs, A_D and A_M must be defined in such a way that they not interfere with the other type of BC, i.e. A_M must return zero on all Dirichlet boundaries so that A_D alone will return the correct value there. And of course, the contribution of $F(\mathbf{x},N)$ would necessarily be defined to contribute nothing to the BCs for all values of $\mathbf{x}$ on the boundary.

6.1 Impact of singularities on the TAS

The task of developing a valid TAS for arbitrary irregular domains was significantly more difficult and complicated than anticipated. This challenge likely led the authors who introduced (47) to abandon their own approach when dealing with irregular boundaries [21]. This chapter follows the progression of research which eventually resulted in a viable TAS. Throughout this development, the existence of singularities in the TAS was a source of continuing frustration. The main focus on the functions composing the TAS was that of imposing requirements at the boundaries of the domain. It was at first believed that the value of the TAS inside the domain was not of significant importance since the ANN could, at least in theory, compensate in order to obtain a good solution. In practice however, singularities inside the domain pose a great threat to fitness of the solution; they require the ANN weights to assume extremes in value, causing problems not only in their vicinity but also farther away as large errors in (5) channeled disproportionate effort into reducing error near them.

The main type of singularity encountered is produced through division by zero. Numerous terms in the TAS require scaling to produce a specific value on boundaries, and zeros can inadvertently appear in the denominator of the scaling factor at some points within the domain. A large amount of time and effort early in this work was devoted to identifying the problematic singularities, determining their source, and understanding their effect on the solution fitness. Only then could a TAS free of singularities be developed. While eventually eliminated from the TAS itself, singularities do still play a role, albeit minor, in the length factors used by the TAS. Singularities could not be eliminated entirely due to the piecewise nature of the boundary definition (introduced in the next section), but are limited to only those points

at boundary segment intersections where the effect on the solution can be minimized. Even so, performance would certainly be improved even further if all singularities could be eliminated through future research.

6.2 Modeling boundary shape and boundary conditions

Representation of the boundary shape and BCs is presented before continuing with the development of the terms A_D, A_M, and F. The boundary of the domain is modeled as a finite number of $D+M$ segments. Segments $1 \le i \le D$ have specified Dirichlet BCs such that the function $g_i(\mathbf{x})$ returns the correct boundary value for any $\mathbf{x}$ on the i^{th} segment. Segments $D < i \le D+M$ have specified Neuman conditions such that $\hat{\mathbf{n}}_i(\mathbf{x}) \cdot \nabla \psi(\mathbf{x}) = g_i(\mathbf{x})$ holds for all $\mathbf{x}$ on the i^{th} segment given that $\hat{\mathbf{n}}_i$ returns the inwardly-directed normal to the segment i. Note that there is no limit on the number of segments (other than being finite) and that each segment may have any arbitrary shape. Also, values for g_i and $\hat{\mathbf{n}}_i$ are not important for $\mathbf{x}$ not on segment i, although monotonic and continuous functions facilitate optimization of ANN parameters.

6.3 Dirichlet contribution to the TAS

In order to tackle a problem of reasonable size, research into the development of a valid TAS began by first considering the special case of only Dirichlet conditions. This section develops functions $A_D(\mathbf{x})$ and $F(\mathbf{x},N)$ for this special case, which later is built upon to introduce Neuman conditions. The functions $A_D(\mathbf{x})$ and $F(\mathbf{x},N)$ introduced here are redefined later by adding more terms to deal with the more general case of mixed BCs.

With the boundary consisting of D Dirichlet boundary segments with corresponding functions $g_i(\mathbf{x})$ for the boundary values, a function of the form

$$A_D(\mathbf{x}) = \sum_{i=1}^{D} \left[g_i(\mathbf{x}) R_i(\mathbf{x}) \right] \tag{60}$$

would be valid if $R_i(\mathbf{x})$ returns unity on boundary segment i and zero on all other segments. Each term in the summation of (60) is designed to return $g_i(\mathbf{x})$ when $\mathbf{x}$ is on boundary i, and not contribution to the value of summation terms for the other boundary segments. Then $A_D(\mathbf{x})$ will always return the correct Dirichlet BCs when $\mathbf{x}$ is on the boundary. The function

$$R_i(\mathbf{x}) \equiv \frac{B_i(\mathbf{x})}{C_i(\mathbf{x})} \tag{61}$$

must then be composed of a function $B_i(\mathbf{x})$ returning zero for all $\mathbf{x}$ on boundaries other than i, and $C_i(\mathbf{x})$ must be defined in such a way that $B_i(\mathbf{x}) = C_i(\mathbf{x}) \neq 0$ for all $\mathbf{x}$ on boundary i.

The TAS will then return correct BCs as long as $F(\mathbf{x},N)$ returns zero for all $\mathbf{x}$ on the domain boundary. Reusing the functions B to define

$$F(\mathbf{x}, N) = N \prod_{i=1}^{D} B_i(\mathbf{x}) \tag{62}$$

produces a complete TAS for the case of only Dirichlet boundaries. Determining valid functions for B and C is all that remains.

6.3.1 Early attempts at defining the TAS

The original concept for defining the functions B and C involved manipulating a function defining the shape of each boundary segment. Each segment was specified by a

function defining its shape, for example a linear boundary segment i in a two-dimensional domain would be defined by a function $b_i(x) = m_i x + a_i$ where m_i and a_i are constants. Any boundary segments which are defined by relations rather than functions can be divided into multiple sub-segments which are functions without loss of generality. With $b_i(\mathbf{x})$ appropriately defined for each segment, the candidate functions

$$B_i(x,y) = \prod_{j=1}^{D}\left[y - b_j(x)\right] \text{ and } C_i(x,y) = \prod_{j=1}^{D}\left[b_i(x) - b_j(x)\right] \tag{63}$$

are valid since $y = b_j(x)$ produces $B_i(x) = 0$ when $\mathbf{x}$ is on boundary j, and $y = b_i(x)$ produces $C_i(x,y) = \prod_{j=1}^{D}\left[y - b_j(x)\right] = B_{ij}(x,y)$ when $\mathbf{x}$ is on boundary i. A TAS using (63) correctly satisfies all Dirichlet conditions.

The values of $R_2(x,y)$ appear in Figure 8 for a triangular domain defined by the intersection of three lines, where the solid white color indicates regions of $|R_2(x,y)|>2$. As desired, the value of $R_2(x,y)$ is unity everywhere along segment 2 and zero along the others. However, notice the singularities at vertical lines through the endpoints of segment 2. These are produced from division by zero when $C_i(x,y)=0$ because $b_2(x)=b_{j=1,3}(x)$ at the intersection of segment 2 segments 1 and 3. Values when approaching one side of the singularities are positive infinity, and values when approaching the other side are negative infinity. ANNs cannot adjust appropriately to handle such extremes in value, and the TAS trained in this manner produces dismal results.

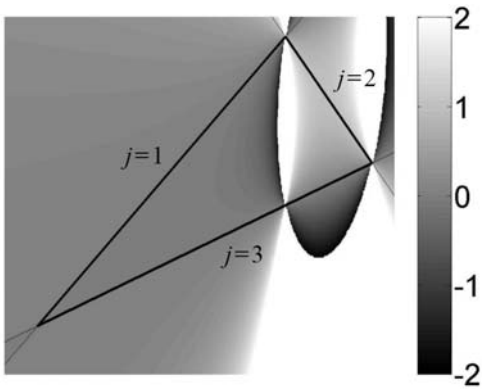


Figure 8. Illustration of $R_2(x,y)$ for a triangular domain using (63)

The functions B and C in (63) compare only y-values of the boundary shapes in b, resulting in the vertical singularities at segment intersection x-values. The x-values of the boundary could also be involved if the inverses of the b functions are available. Again, no generality is lost in requiring inverses since any segment with a function b which is not one-to-one can be broken into sub-segments which are, and thus each sub-segment can be inverted. The candidate functions

$$\begin{aligned} B_i(x,y) &= \prod_{j=1}^{D}\left\{\nu_{ij}\left[x-b_j^{-1}(y)\right]+\upsilon_{ij}\left[y-b_j(x)\right]\right\} \text{ and} \\ C_i(x,y) &= \prod_{j=1}^{D}\left\{\nu_{ij}\left[x-b_j^{-1}(y)\right]+\upsilon_{ij}\left[b_i(x)-b_j(x)\right]\right\} \end{aligned} \tag{64}$$

fulfill the requirements on B and C while incorporating the inverses. The coefficients ν_{ij} and υ_{ij} originally had unity values and were only introduced later after realization that removal of singularities was not guaranteed without them.

Figure 9 illustrates $R_2(x,y)$ using (64) with unity coefficients ν_{ij} and υ_{ij} for the same triangular domain as in Figure 8. Again note unity values along segment 2 and zero along the other two segments. The lines defining the singularities through the two segment endpoints have now been rotated, although one remains within the domain.

Although not obvious by inspection, the function C_i in (64) still has roots which may or may not fall within the domain. Nonetheless, the lines of singularities have been rotated. The inclusion of the coefficients v_{ij} and υ_{ij} provides the possibility of rotating both singularity lines out of the domain. Indeed, coefficient values were chosen by trial and error that produced the function $R_2(x,y)$ in Figure 10 which contains no singularities within the domain boundary.

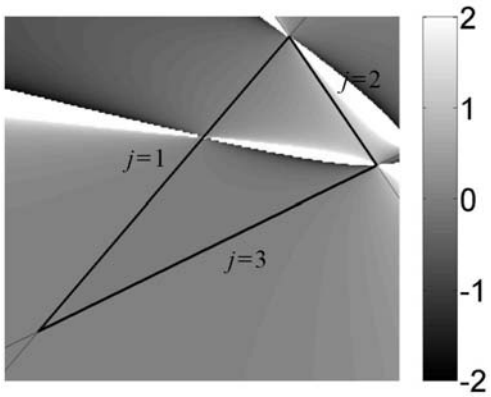


Figure 9. Illustration of $R_2(x,y)$ using (64) with unity coefficients.

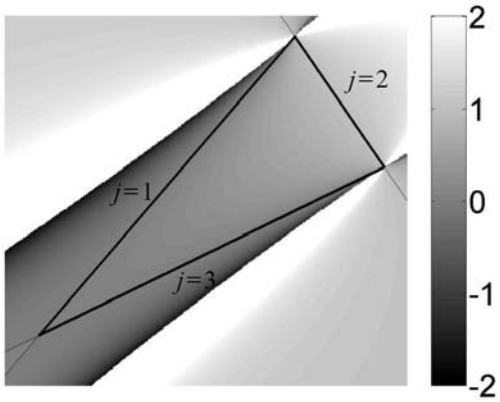


Figure 10. Illustration of $R_2(x,y)$ using (64) with judiciously chosen coefficients.

It became apparent at this point that the C function zeros would never be removed altogether, although they could be rotated outside of the domain. Further

investigation revealed that, at least for linear line segments, the zeros of C could be determined algebraically and thus the coefficients v_{ij} and υ_{ij} determined by solution of a linear system. Solving for these coefficients becomes increasingly more difficult for segments with shapes other than straight lines; in fact, handling even relatively simple quadratic functions is problematic.

Figure 10 illustrates a smooth monotonic function $R_2(x,y)$ within the domain. Similarly defining all of the R_i functions produces a well-behaved TAS satisfying Dirichlet BCs and thus serving as a sound basis for training with an ANN. Note, however, that singularities do still exist at all segment intersections. These singularities are unavoidable since a unity value for R_2 is required along segment 2 simultaneously with a zero value required along intersecting segments 1 and 3. In fact, further investigation of Figure 10 reveals that contours of all values, not simply unity and zero, run through the segment intersections. These singularities do not interfere with training of the solution as long as all training points in T are chosen sufficiently far from boundary segment intersections.

6.3.2 Introduction of length factors

A TAS based on (64), while a great improvement over previous attempts, still involves several drawbacks. As previously mentioned, determining proper coefficient values is difficult for nonlinear boundary shapes. Additionally, the curve defined by the zeros in C will in general not be linear for arbitrary segment shapes; it may be difficult or impossible to rotate the singularities completely out of the domain for more complicated segment shapes. And finally, the effect of B on F in (62) has not yet been discussed. The value of $F(\mathbf{x},N)$ must be zero for all $\mathbf{x}$ on the boundaries, which is

accomplished by (62). However, F will be zero for all (x,y) when $y = b_i(x)$, not just those between the endpoints of the desired segment. For example, consider the domain in Figure 11 composed of three straight lines and a parabola. The solid lines represent the domain boundary, while the dashed lines indicate the continuation of the b_i functions. The point marked in Figure 11 does not lie on a domain boundary, but still has a value $F(\mathbf{x},N)=0$. The TAS is not a function of the ANN output N at this point, and thus the ANN has no control over the TAS's value there.

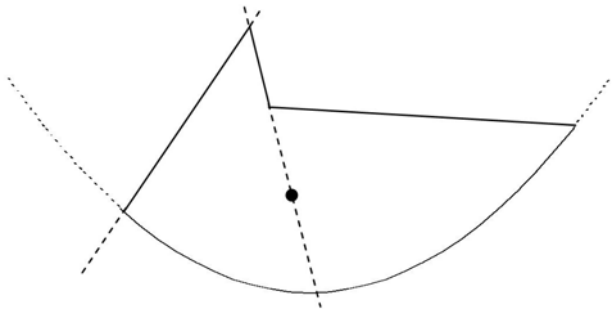


Figure 11. Example domain where (64) is not valid.

A new function R_i was required which could avoid the problems involved with (64). This lead to conceptualization of the length factor, denoted L_j, as a measure of distance from boundary segment j. Length factors can then be used to define

$$B_i = \prod_{j=1,\, j\neq i}^{D} L_j(\mathbf{x}) \text{ and } C_i = \sum_{k=1}^{D}\left[\prod_{j=1,\, j\neq k}^{D} L_j(\mathbf{x})\right] \tag{65}$$

which is used in (61) and then in turn in (60).

The details of the function for the length factor L_j are unimportant as long they fulfill the following two criteria: 1) zero value for all $\mathbf{x}$ on segment j, and 2) non-zero value for all $\mathbf{x}$ within the domain. The first criterion ensures that $\psi_t = g_i(\mathbf{x})$ on segment

i, and the second criterion ensures that the ANN influences the solution everywhere inside the domain. A third requirement will be added later when considering Neuman BCs. Application of (65) produces TAS terms

$$A_D(\mathbf{x}) = \frac{\sum_{i=1}^{D}\left[g_i(\mathbf{x}) \prod_{j=1, j\neq i}^{D} L_j(\mathbf{x}) \right]}{\sum_{i=1}^{D}\left[\prod_{j=1, j\neq i}^{D} L_j(\mathbf{x}) \right]} \text{ and } F(\mathbf{x}, N) = N \prod_{j=1}^{D} L_j(\mathbf{x}) \tag{66}$$

The careful reader will recognize that F in (66) is not entirely consistent with applying (65) to the original definition of F in (62). The consistent result would include an exponent of $D-1$ on the length factors L_j. These exponents are suppresed in (66) since the purpose of F, to return zero on any Dirichlet boundary and be a function of N elsewhere, is served with a unity exponent on each L_j term.

Beyond the two requirements listed above, length factors may be any function, although simple monotonic and continuous functions generally ease the work of optimizing ANN parameters. A length factor as simple as perpendicular distance from the boundary segment is sufficient for many domains (although not for the one in Figure 11 for example). A later chapter investigates actual length factor functions and their effect of TAS performance. The use of length factors in (66) results in a viable TAS automatically satisfying all Dirichlet conditions.

6.4 Extension to Neuman conditions

The term A_M in the TAS

$$\psi_t = A_D + A_M + F \tag{67}$$

must be defined in such a way that the Neuman conditions $\hat{\mathbf{n}}_i(\mathbf{x}) \cdot \nabla \psi_t(\mathbf{x}) = g_i(\mathbf{x})$ are satisfied for all $\mathbf{x}$ on all boundaries $D < i \le D + M$. By definition, Neuman conditions involve first derivatives of the solution, and zero first derivatives of various terms are desirable. For example, the contribution of A_D and F to the first partial derivatives of ψ_t should be zero on Neuman boundaries so as not to contribute to the derivative of the approximate solution already correctly defined by A_M. The strategy employed by [27] when handling Neuman conditions on rectangular domains is to multiply by x^2 any term that must not contribute to the value of the derivative at x=0. This strategy could be employed here together with (66), producing

$$\frac{\partial\left(L_{D+1}^2 F\right)}{\partial x_j} = 2L_{D+1}F + L_{D+1}^2 \frac{\partial F}{\partial x_j} \tag{68}$$

which is zero whenever L_{D+1}=0, that is on Neuman segment D+1. The contribution of F in all partial derivatives could be removed on all Neuman boundaries by multiplying F by the product of all L_i^2 for $D < i \le D + M$. This approach cannot be used, however, since the value as well as the slope of $L_{D+1}^2 F$ is zero on Neuman boundary D+1. In such a case, the *slopes* at Neuman boundaries would be correct automatically (with an appropriately defined A_M), but the ANN would have no control over the *values* there. On the other hand, the strategy in (68) is valid for the A_D term since it need not contribute anything to the value of the TAS on Neuman boundaries.

6.4.1 Attempts to define a valid *F* term

A new strategy is required to define F in order to avoid contributing to $\hat{\mathbf{n}}_i(\mathbf{x})\cdot\nabla\psi_t(\mathbf{x})$ for Neuman boundaries. Functions $f(\mathbf{x})$ other than the quadratic in (68) were considered to force

$$\frac{\partial(fF)}{\partial x_j} = 2\frac{\partial f}{\partial x_j}F + f\frac{\partial F}{\partial x_j} \tag{69}$$

to be zero on Neuman boundaries. Functions such as $\cos(L_{D+1})$ were explored since their derivative is zero for $L_{D+1}=0$ but their values are not. However, such functions could not guarantee the last term in (69) to be zero on boundary D+1. Eventually, it was decided that continued search for a possibly non-existent valid function $f(\mathbf{x})$ did not justify the effort involved, and an alternative approach was adopted.

Forcing (69) to zero simultaneously for all components x_j at Neuman boundaries certainly would produce $\hat{\mathbf{n}}_i\cdot\nabla(fF)=0$, and avoid any influence from F on the overall TAS at Neuman boundaries. However, another approach considers each partial derivative $\partial/\partial x_j$ individually in order to force

$$\sum_{j=1}^{J} n_{ij}\left.\frac{\partial(fF)}{\partial x_j}\right|_{\substack{\text{on Neuman}\\ \text{boundaries}}} = 0 \tag{70}$$

where $n_{ij}(\mathbf{x})$ is the j^{th} component of the inwardly directly normal of the i^{th} boundary segment. The resulting system of differential equations

$$\begin{aligned}
\sum_{j=1}^{J}\left(n_{1j}\frac{\partial f}{\partial x_j}F + n_{1j}f\frac{\partial F}{\partial x_j}\right) &= 0 \\
\sum_{j=1}^{J}\left(n_{2j}\frac{\partial f}{\partial x_j}F + n_{2j}f\frac{\partial F}{\partial x_j}\right) &= 0 \\
&\vdots \\
\sum_{j=1}^{J}\left(n_{Mj}\frac{\partial f}{\partial x_j}F + n_{Mj}f\frac{\partial F}{\partial x_j}\right) &= 0
\end{aligned} \tag{71}$$

is difficult if not impossible to solve, even for the simple case of linear boundary segments where n_{ij} is constant and not a function of position. Obtaining a valid F by solving (71) would produce an aesthetically pleasing TAS where A_D satisfies Dirichlet BCs, A_M satisfies Neuman BCs, and F includes the sole influence of N on the TAS without interfering with BCs satisfied by the other two terms. However, this does not appear to be a promising approach.

The remainder of this chapter emphasizes the importance of finding the simplest possible TAS satisfying the BCs, and departs from compartmentalizing the roles of the three terms A_D, A_M, and F. Distinct roles for the three terms produces a clean and pleasing TAS, but is not a requirement. The TAS must simply satisfy all BCs and be a function of N inside the domain; using three distinct terms was useful for early conception of a valid TAS, but later development departed from this model by the practical necessity for a simple TAS. The TAS resulting from solving (71), if it even exists, would with all likelihood not be as simple as the TAS developed later in this chapter. As a result, further consideration of (71) was abandoned and less complicated methods employed instead.

6.4.2 Removing effects on Neuman conditions from A_D

Even though removing the effect of Neuman conditions from the F term is problematic, the L_i^2 technique can be applied to produce

$$A_D(\mathbf{x}) = \frac{\left(\prod_{j=D+1}^{D+M} L_j^m\right)\sum_{i=1}^{D}\left(g_i \prod_{j=1,j\neq i}^{D} L_j^d\right)}{\sum_{i=1}^{D+M}\left[\left(\prod_{j=1,j\neq i}^{D} L_j^d\right)\left(\prod_{j=D+1,j\neq i}^{D+M} L_j^m\right)\right]} \tag{72}$$

if m=2. The function A_D presented earlier in (66) is simply the special case of (72) with d=1 and m=0. Note that the appropriate boundary value is returned when the length factor for a Dirichlet boundary, segment 1 for example, approaches zero:

$$\begin{aligned}\lim_{L_1=\delta\to 0} A_D(\mathbf{x}) &= \frac{\left(\prod_{j=D+1}^{D+M} L_j^m\right)\left[g_1\prod_{j=2}^{D} L_j^d + \delta^d \sum_{i=2}^{D}\left(g_i \prod_{j=2,j\neq i}^{D} L_j^d\right)\right]}{\left(\prod_{j=2}^{D} L_j^d\right)\left(\prod_{j=D+1}^{D+M} L_j^m\right) + \delta^d \sum_{i=2}^{D+M}\left[\left(\prod_{j=2,j\neq i}^{D} L_j^m\right)\left(\prod_{j=D+1,j\neq i}^{D+M} L_j^2\right)\right]} \\ &= \frac{\left(\prod_{j=D+1}^{D+M} L_j^m\right)\left(g_1\prod_{j=2}^{D} L_j^d\right)}{\left(\prod_{j=2}^{D} L_j^d\right)\left(\prod_{j=D+1}^{D+M} L_j^m\right)} = g_1\end{aligned} \tag{73}$$

The boundary value at the intersection of two Dirichlet segments is

$$\begin{aligned}\lim_{L_1=L_2=\delta\to 0} A_D(\mathbf{x}) &= \frac{\left(\prod_{j=D+1}^{D+M} L_j^m\right)\left[\delta^d g_1\prod_{j=3}^{D} L_j^d + \delta^d g_2\prod_{j=3}^{D} L_j^d + \delta^{2d}\sum_{i=3}^{D}\left(g_i \prod_{j=3,j\neq i}^{D} L_j^d\right)\right]}{2\delta^k\left(\prod_{j=3}^{D} L_j^d\right)\left(\prod_{j=D+1}^{D+M} L_j^m\right) + \delta^d \sum_{i=3}^{D+M}\left[\left(\prod_{j=3,j\neq i}^{D} L_j^d\right)\left(\prod_{j=D+1,j\neq i}^{D+M} L_j^m\right)\right]} \\ &= \frac{\delta^d\left(\prod_{j=D+1}^{D+M} L_j^m\right)\left(\prod_{j=3}^{D} L_j^d\right)(g_1+g_2)}{2\delta^d\left(\prod_{j=2}^{D} L_j^d\right)\left(\prod_{j=D+1}^{D+M} L_j^m\right)} = \frac{g_1+g_2}{2}\end{aligned} \tag{74}$$

which is correct even for a discontinuous change in value from one segment to another.

The intersection of a Dirichlet and Neuman boundary should return the boundary value for the Dirichlet boundary. The limit at such an intersection is

$$\lim_{L_1=L_{D+1}=\delta\to 0} A_D(\mathbf{x}) = \frac{\delta^m\left(\prod_{j=D+2}^{D+M} L_j^m\right)\left[g_1\prod_{j=2}^{D} L_j^d + \delta^d\sum_{i=2}^{D}\left(g_i\prod_{j=2,j\neq i}^{D} L_j^d\right)\right]}{\left(\delta^d+\delta^m\right)\left(\prod_{j=2}^{D} L_j^d\right)\left(\prod_{j=D+2}^{D+M} L_j^2\right)+\delta^3\sum_{i=2,i\neq D+1}^{D+M}\left[\left(\prod_{j=2,j\neq i}^{D} L_j^d\right)\left(\prod_{j=D+2,j\neq i}^{D+M} L_j^m\right)\right]} \quad (75)$$

$$= \frac{\delta^m\left(\prod_{j=D+2}^{D+M} L_j^m\right)\left(g_1\prod_{j=2}^{D} L_j^m\right)}{\left(\delta^d+\delta^m\right)\left(\prod_{j=2}^{D} L_j^d\right)\left(\prod_{j=D+2}^{D+M} L_j^m\right)} = \frac{\delta^m}{\left(\delta^d+\delta^m\right)} g_1$$

which will be zero if $d<m$, $\frac{1}{2}g_1$ if $d=m$, and g_1 if $d>m$. The exponent on the Dirichlet length factors must be larger than that for Neuman factors so that the value at intersections is not forced to an incorrect value. The simplest choices satisfying all requirements are then d=3 and m=2.

6.4.3 Completing Neuman boundary satisfaction with A_M

A new strategy was sought for satisfying Neuman conditions rather than expending the effort to solve (71). The Neuman BC requirements reduced to

$$\hat{\mathbf{n}}_i(\mathbf{x})\cdot\nabla\psi_t(\mathbf{x}) = \hat{\mathbf{n}}_i(\mathbf{x})\cdot\left[\nabla A_D(\mathbf{x})+\nabla A_M(\mathbf{x})+\nabla F(\mathbf{x},N)\right] = \hat{\mathbf{n}}_i(\mathbf{x})\cdot\left[\nabla A_M(\mathbf{x})+\nabla F(\mathbf{x},N)\right]\Big|_{\substack{\text{on Neuman}\\ \text{boundaries } i}} = g_i(\mathbf{x}) \quad (76)$$

when A_D is defined as in (72) with d=3 and m=2. The gradient of F is known explicitly since N and in turn F are known functions of $\mathbf{x}$. Then rather than trying to remove the ∇F term completely, its effect can simply be incorporated into the function A_M. Satisfying Neuman conditions in this manner is simpler, but has the drawback of requiring $A_M(\mathbf{x},N)$ also be a function of N. The basic form

$$A_M = h(\mathbf{x}, N)\prod_{i=1}^{D} L_i \tag{77}$$

was chosen to return zero on all Dirichlet boundaries, and thus not contribute to the value of A_D, and then to properly choose h to satisfy the Neuman conditions.

6.4.4 A valid definition for A_M

The L^2 strategy for removing first derivatives inspired defining the candidate h function

$$h_1(\mathbf{x}, N) = \sum_{i=D+1}^{D+M}\left[g_i'(\mathbf{x}, N) L_i \prod_{j=D+1, j\neq i}^{D+M} L_j^2 \right] \tag{78}$$

which utilizes a summation where each g_i' term will be defined to satisfy Neuman conditions for the i^{th} boundary. The first partial derivative of (78) with respect to the components of $\mathbf{x}$ is

$$\begin{aligned}\frac{\partial h_1}{\partial x_k} &= \frac{\partial\left(g_i' L_i \prod_{j=D+1, j\neq i}^{D+M} L_j^2 \right)}{\partial x_k} + \frac{\partial\left[L_i^2 \sum_{j=D+1, j\neq i}^{D+M}\left(g_j' L_j \prod_{l=D+1, l\neq i, j}^{D+M} L_l^2 \right)\right]}{\partial x_k} = \frac{\partial\left(g_i' L_i \prod_{j=D+1, j\neq i}^{D+M} L_j^2 \right)}{\partial x_k}\Bigg|_{L_i=0} \\ &= L_i \frac{\partial\left(g_i' \prod_{j=D+1, j\neq i}^{D+M} L_j^2 \right)}{\partial x_k} + \frac{\partial L_i}{\partial x_k} g_i' \prod_{j=D+1, i\neq i}^{D+M} L_j^2 = \frac{\partial L_i}{\partial x_k} g_i' \prod_{j=D+1, j\neq i}^{D+M} L_j^2\end{aligned} \tag{79}$$

when evaluated on Neuman boundary segment i. The $L_{j\neq i}^2$ factors in (78) ensure that only the g_i' term affects the derivative in (79), and the L_i factor in (78) ensures that (79) is an algebraic and not differential equation in g_i'. Equations (76) through (79) are then combined to produce

$$\sum_{j=1}^{J} n_{ij}\frac{\partial F}{\partial x_j} + \sum_{j=1}^{J} n_{ij}\frac{\partial A_M}{\partial x_j} = \sum_{j=1}^{J} n_{ij}\frac{\partial F}{\partial x_j} + g_i'\left(\prod_{j=1}^{D} L_j\right)\left(\prod_{j=D+1, j\neq i}^{D+M} L_j^2\right)\left(\sum_{j=1}^{J} n_{ij}\frac{\partial L_i}{\partial x_j}\right) = g_i\Big|_{L_{i>D}=0} \tag{80}$$

which is rearranged to generate

$$g_i' = \left. \frac{g_i - \sum_{j=1}^{J} n_{ij} \frac{\partial F}{\partial x_j}}{\left(\prod_{j=1}^{D} L_j \right) \left(\prod_{j=D+1, j\neq i}^{D+M} L_j^2 \right) \sum_{j=1}^{J} \left(n_{ij} \frac{\partial L_i}{\partial x_j} \right)} \right|_{L_{i>D}=0} \tag{81}$$

when using $h = h_1$. The function g_i' must have the value specified in (81) when evaluated at **x** on boundary *i*; the value of g_i' is not important for other **x**. The general form

$$g_i' = \frac{g_i - \sum_{j=1}^{J} n_{ij} R_j(\mathbf{x}, N)}{\left(\prod_{j=1}^{D} L_j^{\gamma} \right) \left(\prod_{j=D+1, j\neq i}^{D+M} L_j^{\mu} \right) \left(\sum_{j=1}^{J} n_{ij} \frac{\partial L_i}{\partial x_j} \right) + P_i(\mathbf{x})} \tag{82}$$

is introduced which has (81) as a special case. Note that the prime in g_i' indicates an alternate g_i function and not a derivative. The special case in (81) cannot be used as the complete function for g_i' since the denominator would be zero on any boundary other than *i*. A function for $P_i(\mathbf{x})$ must be chosen which is zero on segment *i* and non-zero on all other segments. The obvious choice of $P_i(\mathbf{x}) = L_i(\mathbf{x})$ removes singularities along the i^{th} segment, but does not eliminate them at segment endpoints. The function

$$P_i(\mathbf{x}) = \zeta \frac{L_i (L_a + L_b)}{L_a L_b + L_i (L_a + L_b)} \tag{83}$$

is then introduced where *a* and *b* are chosen to be the two segments bordering segment *i*, and ζ is a scaling factor with a unity default value. Evaluating (83) on segment *i* is

$$\lim_{L_i=\delta\to 0} P_i(\mathbf{x}) = \zeta \frac{\delta}{L_a L_b + \delta} = 0 \tag{84}$$

as required and evaluating at the intersection of *i* and *a* is

$$\lim_{L_i=L_a=\delta\to 0} P_i(\mathbf{x}) = \zeta \frac{\delta(\delta+L_b)}{\delta L_b + \delta(\delta+L_b)} = \zeta \frac{\delta L_b}{2\delta L_b} = \frac{1}{2}\zeta \neq 0 \tag{85}$$

Since length factors are always positive and non-zero inside the domain, (83) satisfies all requirements while eliminating most singularities in the denominator of (82). The result is a valid definition for A_M using (77), $h = h_1$ in (78), (82), (83), $\gamma = 1$, $\mu = 2$, and $R_j(\mathbf{x}, N) = \partial F / \partial x_j$. The only problem with this definition for A_M lies in possible zero denominators in (82) resulting from the product of n_{ij} and $\partial L_i / \partial x_j$ as they are the only terms which can be negative. This does not present a problem in most cases since length factors increase in general when moving away from a segment and will thus both n_{ij} and $\partial L_i / \partial x_j$ will have the same sign on the interior side of segment *i*. Severely concave domain shapes could produce a negative $n_{ij}\partial L_i / \partial x_j$ product, requiring an increase of the scaling factor ζ in compensation. More details on this topic appear in Chapter 7 where actual length factor functions are considered.

6.5 A complete TAS for mixed conditions

The previous section extended to treatment of mixed BCs, resulting in the TAS

$$\psi_t = F + A_D + A_M = N\prod_{j=1}^{D} L_j + \frac{\left(\prod_{j=D+1}^{D+M} L_j^2\right)\sum_{i=1}^{D}\left(g_i \prod_{j=1,\,j\neq i}^{D} L_j^3\right)}{\sum_{i=1}^{D+M}\left[\left(\prod_{j=1,\,j\neq i}^{D} L_j^3\right)\left(\prod_{j=D+1,\,j\neq i}^{D+M} L_j^2\right)\right]} + \left(\prod_{j=1}^{D} L_j\right)\sum_{i=D+1}^{D+M}\left[\frac{L_i\left(\prod_{j=D+1,\,j\neq i}^{D+M} L_j^2\right)\left(g_i - \sum_{j=1}^{J} n_{ij}\frac{\partial F}{\partial x_j}\right)}{\left(\prod_{j=1}^{D} L_j\right)\left(\prod_{j=D+1,\,j\neq i}^{D+M} L_j^2\right)\left(\sum_{j=1}^{J} n_{ij}\frac{\partial L_i}{\partial x_j}\right) + \zeta\frac{L_i(L_a+L_b)}{L_a L_b + L_i(L_a+L_b)}}\right] \tag{86}$$

when using (78) to define A_M. This TAS automatically satisfies all BCs, both Dirichlet and Neuman, and is free of singularities within the domain, along boundaries, and even at boundary segment intersections (this of course assuming that length factor definitions do not introduce any singularities).

This TAS has been demonstrated to successfully solve several BVPs with Dirichlet or mixed BCs. Choosing correct design parameters, i.e. the number of hidden nodes H and the number and location of training points for the set T, became more difficult when the number of boundary segments in the domain increased. BVPs with up to five boundary segments could be solved successfully, but a problem with twelve segments could not be solved regardless of design parameter choice. After much investigation, it became apparent that although never zero, the denominators of terms in (86) were so small that they behaved like singularities in their ability to frustrate efforts of the ANN to obtain locally accurate solutions. The length factor values were in general less than unity (they are required to be zero on boundaries) and stringing multiple products of squared and/or cubed length factors together could easily produce very small values. A different TAS with smaller length factor exponents was required to solve problems with a number of boundary segments of practical interest.

6.5.1 Reducing exponents on length factor terms in A_D

The squared length factors in A_D in (86) were included in order to force $\partial A_D / \partial x_j = 0$ on Neuman boundaries and thus not interfere with A_M by contributing to $\hat{\mathbf{n}}_i(\mathbf{x}) \cdot \nabla \psi_t(\mathbf{x})$ for Neuman conditions. And the cubic length factors in (86) are required to prevent the squared ones from creating incorrect solution values near segment intersections. Remember that unlike A_D, no appropriate function was found for F so that

its contribution to Neuman conditions could be eliminated. This problem was solved by incorporating the contribution of F on the first derivatives into the definition of A_M. The same could be accomplished simply for A_D by redefining

$$R_j\left(\mathbf{x},N\right)=\frac{\partial A_D}{\partial x_j}+\frac{\partial F}{\partial x_j} \tag{87}$$

to be used in (82). The original definition of A_D with d=1 and m=0 in (72) could be used to significantly simplify the TAS to

$$\psi_t = F + A_D + A_M = N\prod_{j=1}^{D} L_j + \frac{\sum_{i=1}^{D}\left(g_i \prod_{j=1,\, j\neq i}^{D} L_j\right)}{\sum_{i=1}^{D}\left[\left(\prod_{j=1,\, j\neq i}^{D} L_j\right)\right]} + \left(\prod_{j=1}^{D} L_j\right)\sum_{i=D+1}^{D+M}\left\{\frac{L_i\left(\prod_{j=D+1,\, j\neq i}^{D+M} L_j^2\right)\left[g_i-\sum_{j=1}^{J} n_{ij}\left(\frac{\partial A_D}{\partial x_j}+\frac{\partial F}{\partial x_j}\right)\right]}{\left(\prod_{j=1}^{D} L_j\right)\left(\prod_{j=D+1,\, j\neq i}^{D+M} L_j^2\right)\left(\sum_{j=1}^{J} n_{ij}\frac{\partial L_i}{\partial x_j}\right)+\zeta\frac{L_i\left(L_a+L_b\right)}{L_aL_b+L_i\left(L_a+L_b\right)}}\right\} \tag{88}$$

without affecting the automatic satisfaction of BCs. This reduction in the length factor exponents in A_D led to consideration of alternative definitions of A_M to reduce exponents there as well.

6.5.2 Reducing exponents on length factor terms in A_M

A more general function for h in (77) was sought which hopefully would reduce length factor exponents. A function of the form

$$h\left(\mathbf{x},N\right)=h_2\left(\mathbf{x},N\right)\prod_{j=D+1}^{D+M} L_j \tag{89}$$

was considered since the L_j terms, in a similar manner as in (79), would produce an algebraic rather than a differential equation to solve in terms of h_2. The Neuman condition requirement using (89) is

$$\sum_{j=1}^{J} n_{ij}\left(\frac{\partial A_D}{\partial x_j}+\frac{\partial F}{\partial x_j}\right)+h_2\left(\prod_{j=1}^{D} L_j\right)\left(\prod_{j=D+1}^{D+M} L_j\right)\left(\sum_{j=1}^{J} n_{ij}\frac{\partial L_i}{\partial x_j}\right)=g_i\Bigg|_{L_{i>D}=0} \tag{90}$$

Note that $\partial L_i/\partial x_j$ may not equal zero for all j at any location on segment i if the Neuman BC is to be satisfied; this represents the *third criterion* for length factor definition: at least one first partial derivative of length factor i must be non-zero for all $\mathbf{x}$ on boundary i. The requirement on h_2 is determined by rearranging (90) to obtain

$$h_2\left(\mathbf{x},N\right)\Big|_{L_{i>D}=0}=\frac{g_i-\sum_{j=1}^{J} n_{ij}\left(\frac{\partial A_D}{\partial x_j}+\frac{\partial F}{\partial x_j}\right)}{\left(\prod_{j=1,\,j\neq i}^{D+M} L_j\right)\left(\sum_{j=1}^{J} n_{ij}\frac{\partial L_i}{\partial x_j}\right)} \tag{91}$$

Recognize now that (91) indicates that h_2 must assume a particular functional value at each of the $D<i\le D+M$ boundaries, and that functional value is a special case of g_i' from (82). The more general

$$h_2\left(\mathbf{x},N\right)\Big|_{L_{i>D}=0}=\frac{g_i-\sum_{j=1}^{J} n_{ij}\left(\frac{\partial A_D}{\partial x_j}+\frac{\partial F}{\partial x_j}\right)}{\left(\prod_{j=1,\,j\neq i}^{D+M} L_j\right)\left(\sum_{j=1}^{J} n_{ij}\frac{\partial L_i}{\partial x_j}\right)+\zeta\frac{L_i\left(L_a+L_b\right)}{L_aL_b+L_i\left(L_a+L_b\right)}}=g_i' \tag{92}$$

is required to avoid singularities in the denominator of (91). The problem of forcing h_2 to return g_i' at each of the i Neuman boundaries is the same as requiring A_D to return g_i on all Dirichlet boundaries. A definition for h_2 analogous to (66) is then

$$h_2\left(\mathbf{x},N\right)=\frac{\sum_{i=D+1}^{D+M}\left(g_i'\prod_{j=D+1,\,j\neq i}^{D+M}L_j\right)}{\sum_{i=D+1}^{D+M}\left(\prod_{j=D+1,\,j\neq i}^{D+M}L_j\right)} \tag{93}$$

which combined with (77), (89), and (92) generates

$$A_M=\frac{\prod_{j=1}^{D+M}L_j}{\sum_{i=D+1}^{D+M}\left(\prod_{j=D+1,\,j\neq i}^{D+M}L_j\right)}\sum_{i=D+1}^{D+M}\left\{\frac{\left(\prod_{j=D+1,\,j\neq i}^{D+M}L_j\right)\left[g_i-\sum_{j=1}^{J}n_{ij}\left(\frac{\partial A_D}{\partial x_j}+\frac{\partial F}{\partial x_j}\right)\right]}{\left(\prod_{j=1,\,j\neq i}^{D+M}L_j\right)\left(\sum_{j=1}^{J}n_{ij}\frac{\partial L_i}{\partial x_j}\right)+\zeta\frac{L_i\left(L_a+L_b\right)}{L_aL_b+L_i\left(L_a+L_b\right)}}\right\} \tag{94}$$

This definition of A_M removes the quadratic exponents appearing in (88), and the resulting TAS contains no length factor exponents greater than unity. This TAS has successfully solved all BVPs tested to date.

6.5.3 The TAS at Neuman segment intersections

The TAS at the intersection of two Dirichlet segments was shown to correctly return the average of the two segment BCs in (74). Verifying the solution at the intersection of Neuman segments is of equal importance. In cases where two Neuman segments intersect, the limit

$$\lim_{L_{D+1}=L_{D+2}=\delta\to 0}A_M=\delta^2\left(\prod_{i=1,i\neq D+1,D+2}^{D+M}L_i\right)\left(\frac{g'\big|_{L_{D+1}=0}+g'\big|_{L_{D+2}=0}}{2}\right)=0 \tag{95}$$

will be zero for the everywhere-finite g' from (92). The ANN will be trained to generate a proper *value* on the Neuman boundary through adjustment of the F term. Computing the gradient

$$\frac{\partial A_M}{\partial x_k} = \frac{\partial\left[L_{D+1}L_{D+2}\left(\prod_{j=1,i\neq D+1,D+2}^{D+M} L_j\right)h_2\right]}{\partial x_k} = \frac{\partial L_{D+1}}{\partial x_k} L_{D+2}\left(\prod_{j=1,i\neq D+1,D+2}^{D+M} L_j\right)h_2 + L_{D+1}\frac{\partial\left[L_{D+2}\left(\prod_{j=1,i\neq D+1,D+2}^{D+M} L_j\right)h_2\right]}{\partial x_k} \tag{96}$$

and evaluating at the intersection of segments D+1 and D+2 reveals that

$$\lim_{L_{D+1}=L_{D+2}=\delta\to 0}\frac{\partial A_M}{\partial x_k} = \delta\frac{\partial L_{D+1}}{\partial x_k}\left(\prod_{j=1,i\neq D+1,D+2}^{D+M} L_j\right)h_2 + \delta\frac{\partial\left[L_{D+2}\left(\prod_{j=1,i\neq D+1,D+2}^{D+M} L_j\right)h_2\right]}{\partial x_k} = 0 \tag{97}$$

does not necessarily produce the desired average of Neuman conditions:

$$\lim_{L_{D+1}=L_{D+2}=\delta\to 0}\hat{\mathbf{n}}_i\cdot\nabla\psi_t = \sum_{j=1}^{J} n_{ij}\left(\frac{\partial A_D}{\partial x_j}+\frac{\partial A_M}{\partial x_j}+\frac{\partial F}{\partial x_j}\right) = \sum_{j=1}^{J} n_{ij}\left(\frac{\partial A_D}{\partial x_j}+\frac{\partial F}{\partial x_j}\right) \tag{98}$$

Equation (98) does not equal the desired average of g_{D+1} and g_{D+2}. This problem, of course, is only encountered at segment intersections, and $\partial F/\partial x_j$ is still a function of the ANN output which can compensate by adjusting itself appropriately.

Another case of segment intersection is between Dirichlet and Neuman segments. Dirichlet BCs are not affected by consideration of Neuman conditions since the A_D definition in (66) contains no $L_{i>D}$ terms. The Neuman condition, however, is not satisfied at these intersections with an argument similar to (97) and (98) when L_1 and L_{D+1} approach zero. However, this concern is likewise addressed by the ability of the ANN to adjust $\partial F/\partial x_j$.

The previous statement that all BCs are automatically satisfied everywhere along the boundary must be qualified with the exception of Neuman BCs in the limit

approaching Neuman segment endpoints. Neuman BCs are automatically satisfied immediately surrounding the endpoints, and the ANN can adjust for any inconsistencies precisely at endpoints.

6.6 Summary of TAS development

This chapter has illustrated that a TAS automatically satisfying all BCs is far from unique. A long line of candidate TASs were considered, and although each could correctly satisfy the BCs, they were rejected for practical reasons and modified until

$$\psi_t = F + A_D + A_M = N\prod_{j=1}^{D} L_j + \frac{\sum_{i=1}^{D}\left(g_i \prod_{j=1,\, j\neq i}^{D} L_j \right)}{\sum_{i=1}^{D}\left[\left(\prod_{j=1,\, j\neq i}^{D} L_j \right) \right]} + \frac{\prod_{j=1}^{D+M} L_j}{\sum_{i=D+1}^{D+M}\left(\prod_{j=D+1,\, j\neq i}^{D+M} L_j \right)} \sum_{i=D+1}^{D+M} \left\{ \frac{\left(\prod_{j=D+1,\, j\neq i}^{D+M} L_j \right)\left[g_i - \sum_{j=1}^{J} n_{ij}\left(\frac{\partial A_D}{\partial x_j} + \frac{\partial F}{\partial x_j} \right) \right]}{\left(\prod_{j=1,\, j\neq i}^{D+M} L_j \right)\left(\sum_{j=1}^{J} n_{ij} \frac{\partial L_i}{\partial x_j} \right) + \zeta \frac{L_i\left(L_a + L_b\right)}{L_a L_b + L_i\left(L_a + L_b\right)}} \right\} \tag{99}$$

was developed.

The elimination of singularities in the domain, on boundaries, and at segment intersections was the prevalent practical concern driving evolution of the TAS definition. Formation of the length factor concept, although simple and intuitive, represented a major innovation in developing a singularity-free TAS. The final result in (99) is in itself free of singularities, even for BCs discontinuous in value, although requirements on length factor definition often do introduce some relatively minor singularities, especially at segment intersections, whose effect can be neutralized through judicious choice of design parameters.

A second practical issue when defining the TAS proved to be finding simpler solutions with smaller exponents on length factor terms. High exponents force terms in the TAS to be large (with length factor values above unity) at some locations in the domain and small (with length factor values below unity) at others. The dynamic range of ANNs is not sufficient to handle such extremes in values, especially for domains consisting of numerous boundary segments, forcing the development of simpler TASs. While (99) has been shown to adequately solve all BVPs considered to date, the possibility exists that even simpler, and thus more computationally effective, TASs exist.

6.7 Preparation of TAS partial derivatives for training

Optimization by gradient descent requires the partial derivative $\partial E/\partial \boldsymbol{\theta}$ be computed. The TAS in (99) automatically satisfies all BCs and so the error function in (5) can be used where the function G is determined by the DE as in (58). The resulting error gradient is

$$\frac{\partial E}{\partial \boldsymbol{\theta}} = \frac{2}{|T|} \sum_{\mathbf{x} \in T} \left(G \frac{\partial G}{\partial \boldsymbol{\theta}} \right) \tag{100}$$

Whatever partial derivatives compose the DE will be differentiated with respect to the ANN weights $\boldsymbol{\theta}$. For example, solving the Laplace equation would require computation of $\partial^3 \psi_t / \partial x_j^2 \partial \boldsymbol{\theta}$ for each x_j component of $\mathbf{x}$. The function ψ_t is of course dependent on the ANN output; all the relevant partial derivatives of N were developed in (43) through (45). What remains then is to develop expressions for the partial derivatives of ψ_t with respect to the components x_j. For the Laplace equation example, this obviously requires $\partial^2 A_D / \partial x_j^2$, $\partial^2 A_M / \partial x_j^2$, and $\partial^2 F / \partial x_j^2$. Less obvious is that the partial

derivatives of A_D and F are included in the definition of A_M in (99). This requires expressions for $\partial^3 A_D / \partial x_k \partial x_j^2$ and $\partial^3 F / \partial x_k \partial x_j^2$ even for the Laplace equation example. Note also that partial derivatives of the boundary functions $g_i(\mathbf{x})$, boundary shapes in $n_{ij}(\mathbf{x})$, and length factors $L_i(\mathbf{x})$ must also be available.

The terms composing (99) involve functions no more complicated that addition, subtraction, multiplication, and division. Most notably, products of multiple length factors are prevalent. When preparing derivatives of these products, consider the general case of the function

$$g(x_1, x_2, \ldots, x_m) = \prod_{i=1}^{m} f_i(x_1, x_2, \ldots, x_m) \tag{101}$$

This function has first, second, and third partial derivatives

$$\frac{\partial g}{\partial x_n} = \sum_{i=1}^{m} \left(\frac{\partial f_i}{\partial x_n} \prod_{j=1, j\neq i}^{m} f_j \right), \tag{102}$$

$$\frac{\partial^2 g}{\partial x_n^2} = \sum_{i=1}^{m} \left[\frac{\partial^2 f_i}{\partial x_n^2} \prod_{j=1, j\neq i}^{m} f_j + 2 \frac{\partial f_i}{\partial x_n} \sum_{j>i}^{M} \left(\frac{\partial f_j}{\partial x_n} \prod_{k=1, k\neq i, j}^{m} f_k \right) \right], \text{ and} \tag{103}$$

$$\frac{\partial^3 g}{\partial x_n^3} = \sum_{i=1}^{m} \left\{ \begin{array}{l} \dfrac{\partial^3 f_i}{\partial x_n^3} \displaystyle\prod_{j=1, k\neq i}^{m} f_j + 3 \frac{\partial^2 f_i}{\partial x_n^2} \sum_{j>i}^{m} \left(\frac{\partial f_j}{\partial x_n} \prod_{k=1, k\neq i, j}^{m} f_k \right) + \\ 6 \dfrac{\partial f_i}{\partial x_n} \displaystyle\sum_{j>i}^{m} \left[\frac{\partial f_j}{\partial x_n} \sum_{k>j}^{m} \left(\frac{\partial f_k}{\partial x_n} \prod_{l=1, l\neq i, j, k}^{m} f_l \right) \right] \end{array} \right\} \tag{104}$$

respectively.

Developing explicit expressions for the requisite mixed partial derivatives of ψ_t is straightforward yet time-consuming, using basic tools such as the chain rule and the product and quotient rules. The laborious details are omitted here since their addition

would add no real value to this work. The resulting error gradient $\partial E / \partial \mathbf{\theta}$ then serves as a basis for any gradient descent optimization approach, such as the RPROP and Levenberg-Marquardt methods presented earlier.

7 Length factor definition

Length factor definitions for $L_i(\mathbf{x})$ must fulfill three criteria: 1) zero value for all **x** on segment *i*, 2) non-zero value for all **x** within the domain, and 3) at least one non-zero partial derivative $\partial L_i / \partial x_j$ everywhere on segment *i*. The simplest imaginable length factor is the perpendicular distance from a point **x** of interest within the domain to the curve defining each boundary segment *i*, as illustrated in Figure 12. Perpendicular distance satisfies all three criteria as long as the curves defining segment boundaries do not enter the domain as they do in Figure 11. Perpendicular distance in this later case would violate the second criterion requiring non-zero length factors inside the domain.

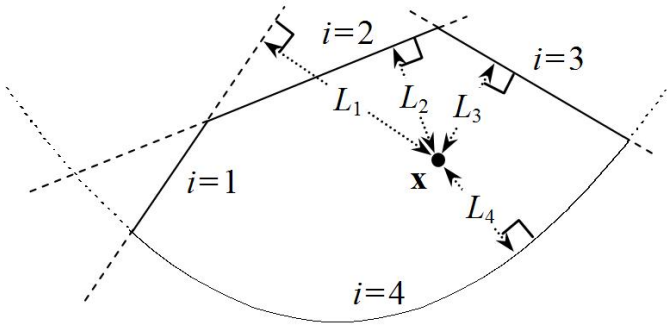


Figure 12. Illustration of perpendicular distance as a length factor.

7.1 Length factor definition for linear boundary segments

To illustrate how a length factor can be defined to avoid the problem with perpendicular distance when a segment boundary enters the domain, consider the case of a linear boundary segment defined by *ac* in Figure 13, where the length factor is to be calculated relative to point *d*. Subtracting $\overline{ac}$ from $\overline{ad}+\overline{cd}$ would return zero if and only if *d* falls on segment *ac*, thus satisfying the first two length factor criteria. The third

criterion would not be satisfied however since the partial derivatives of $\overline{ad}$ and $\overline{cd}$ would be equal and opposite when *d* lies on *ac*, resulting in zero for all $\partial L/\partial x_k$ since $\overline{ac}$ is constant. An appropriate length factor is then defined as

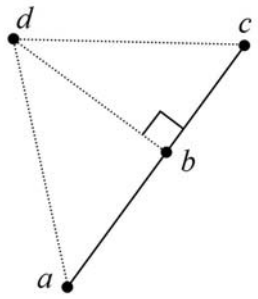


Figure 13. Geometry for developing a satisfactory length factor.

$$L = \alpha\left(\overline{ad} + \overline{cd} - \overline{ac} + \xi\overline{bd}\right) \tag{105}$$

so that all three criteria are upheld by adding $\overline{bd}$ to ensure that the third criterion is satisfied as well. The scaling factor ξ in (105) has a unity default value.

The constant α in (105) simply scales the magnitude of *L*. The factor α is important for two reasons. The products of length factors in the TAS in (99) can saturate the ANN with exceedingly large values if length factors are allowed to be significantly larger than unity. Additionally, the second partial derivatives $\partial^2 L/\partial x_k^2$ are necessarily infinite at points *a* and *c* if the first two length factor requirements are fulfilled. Evaluating $\widetilde{D}\psi_t(\mathbf{x}, N) - f(\mathbf{x})$ at boundary intersections *a* and *c* for any (1) involving second partial derivatives will produce an infinite result. This does not present a significant problem for optimization of the ANN as long as the large second derivative values are confined to a region close to the intersection points. Scaling with a value of α

less than unity will reduce the value of L, and its derivatives as well. This reduces the size of the region around the intersection influenced by the singularity.

Care must be taken, however, that α is not chosen too small. Remember that exponents d and m greater than unity on length factors L_i^d and L_i^m were eliminated from the TAS so that multiple products of these terms did not produce exceedingly small values with length factors less than unity. Choosing α too small would have the same deleterious effect, and so the value for α must be carefully chosen.

Figure 14 illustrates length factor contours for a horizontal segment of unit length where the perpendicular distance $\overline{bd}$ from (105) is positive on both sides of the segment. This length factor can be used for a domain with any concavity since the only zero contour is the segment itself. Notice, however, that the first partial derivative with respect to y is discontinuous along the x-axis due to the sharp point in the eye-shaped contours. This results from a positive distance $\overline{bd}$ on both the top and bottom sides of the segment.

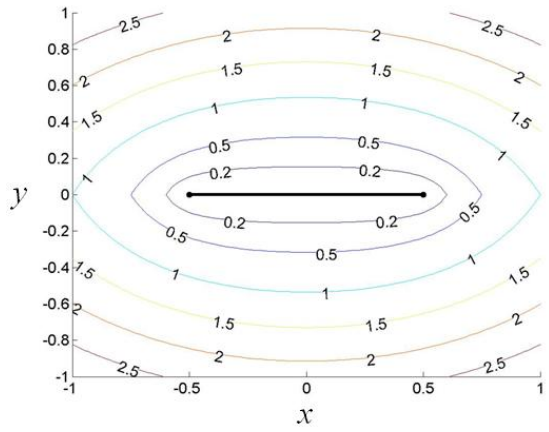


Figure 14. Contours of constant length factor using (105) with a positive $\overline{bd}$ and ξ=1.

The discontinuity in derivative apparent in Figure 14 can be avoided if $\overline{bd}$ is instead defined to be positive on the side of the segment "inside" the domain and negative on the side "outside" the domain. The length factor contours in Figure 15 are free of discontinuities by using a signed $\overline{bd}$. Length factor values are positive above the segment, where interior of the domain is assumed to lie, but negative in a relatively small region below the segment. The region of negative length factor value can be minimized by decreasing the scaling factor ξ as apparent in the comparison of Figure 15a and 15b with ξ values of 1 and ½ respectively. The factor ξ could be reduced further in the unlikely case of severely concave domains.

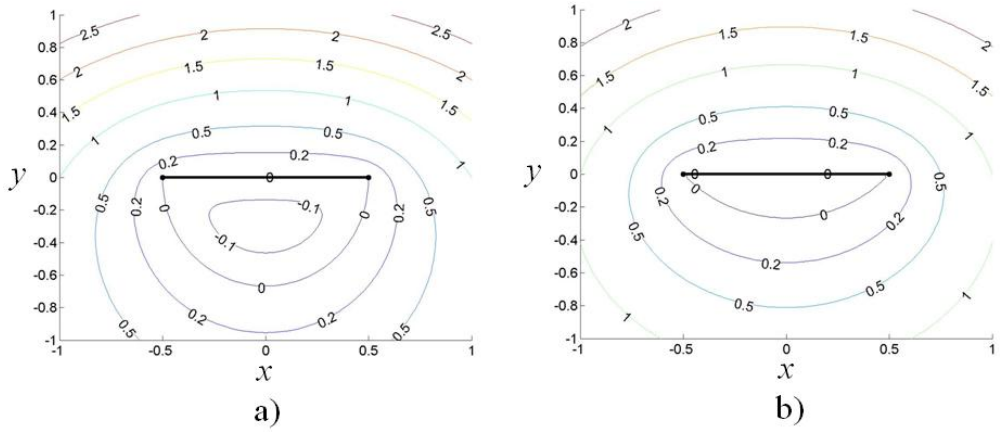


Figure 15. Contours of constant length factor using (105) with a signed $\overline{bd}$ and a) ξ=1 and b) ξ=½.

The scaling factor ξ cannot be reduced to zero since $\overline{bd}$ appears in (105) to guarantee a non-zero partial derivative $\partial L_i / \partial x_j$ on segment *i*. While reducing ξ does not jeopardize the third requirement on length factor definition, it does affect possible singularities in A_M resulting from zero denominators as introduced in Chapter 6.4.4. The denominator of A_M (99) will be negative if the signs of n_{ij} and $\partial L_i / \partial x_j$ are opposite and

their magnitudes large enough to produce a negative result after adding the positive value of $P_i(\mathbf{x})$. The value of n_{ij} is the j^{th} component of the inwardly directed normal, and will be a constant value for linear segments. It is thus important to identify locations within the domain which have negative values of $\partial L_i / \partial x_j$.

Figure 16 illustrates contours of constant $\partial L/\partial y$ for a horizontal segment of unit length using $\xi = 1$ in (105). Similar to the value of L, $\partial L/\partial y$ also exhibits a negative region "behind" the boundary segment. This region, however, is larger than for the value of L, and could possibly cause the denominator of A_M to be zero. Additionally, lowering ξ to reduce the size of the negative region in L will in fact increase the size of the negative region in $\partial L/\partial y$. Notice that $\partial \overline{bd}/\partial y = 1$ for a segment of this orientation. Removing $\overline{bd}$ entirely with $\xi = 0$ would reduce $\partial L/\partial y$ by exactly one everywhere in the domain. The contour lines in Figure 16 would have the same shapes for $\xi = 0$ but the values would be shifted so that horizontal contour at $y = 0$ would have a zero value; reducing ξ effectively increases the size of the negative region in $\partial L/\partial y$.

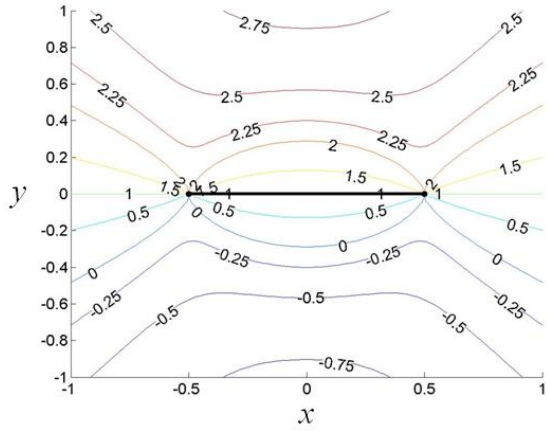


Figure 16. Contours of constant $\partial L/\partial y$ using (105) with a signed $\overline{bd}$ and ξ=1.

Existence of negative products $n_{ij}\,\partial L_i/\partial x_j$ does not necessarily translate to a zero A_M denominator

$$\left(\prod_{j=1,\,j\neq i}^{D+M} L_j\right)\left(\sum_{j=1}^{J} n_{ij}\frac{\partial L_i}{\partial x_j}\right)+\zeta\frac{L_i\left(L_a+L_b\right)}{L_aL_b+L_i\left(L_a+L_b\right)} \tag{106}$$

since the second term is always positive. Remember that *a* and *b* indicate the segments bordering segment *i*, and ζ is a scaling factor with a unity default value. The product of length factors in the first term of (106) will generally be smaller than unity for the region in question since large products are only possible far away from all segments, i.e. in the central region of the domain. The magnitude of the product $n_{ij}\,\partial L_i/\partial x_j$ will thus in general be reduced before addition with the positive second term. The second term was designed specifically to return zero on segment *i*, but will return values between $\frac{1}{2}\zeta$ and ζ for all locations reasonably far from the center of segment *i* given the limits

$$\lim_{L_a \text{ and/or } L_b\to 0}\zeta\frac{L_i\left(L_a+L_b\right)}{L_aL_b+L_i\left(L_a+L_b\right)}=\zeta\,, \tag{107}$$

$$\lim_{L_i \text{ and } (L_a \text{ or } L_b)\to 0}\zeta\frac{L_i\left(L_a+L_b\right)}{L_aL_b+L_i\left(L_a+L_b\right)}=\frac{1}{2}\zeta\,,\text{ and} \tag{108}$$

$$\lim_{L_a, L_b \text{ and } L_i\to\infty}\zeta\frac{L_i\left(L_a+L_b\right)}{L_aL_b+L_i\left(L_a+L_b\right)}=\frac{2}{3}\zeta \tag{109}$$

Figure 17 illustrates values of a) $\partial L/\partial y$ and b) the denominator of A_M from (106) for a horizontal boundary segment on a concave domain investigated in the next chapter. Figure 17a displays the same information as Figure 16 except that values outside of the star-shaped domain are suppressed and the interior of the domain lies below the boundary segment. A small region near the top point of the star does have negative

$\partial L/\partial y$ values, but the second term in (106) compensates to avoid zero denominators as evidenced by Figure 17b. More severely concave domains could still result in zero denominators, although ζ can be increased in such a case to compensate. In fact, ζ need not be constant; it could be any function as long as its value is non-zero at segment i intersections. Unity or somewhat larger values of ζ will, however, be sufficient for the vast majority of domains in question.

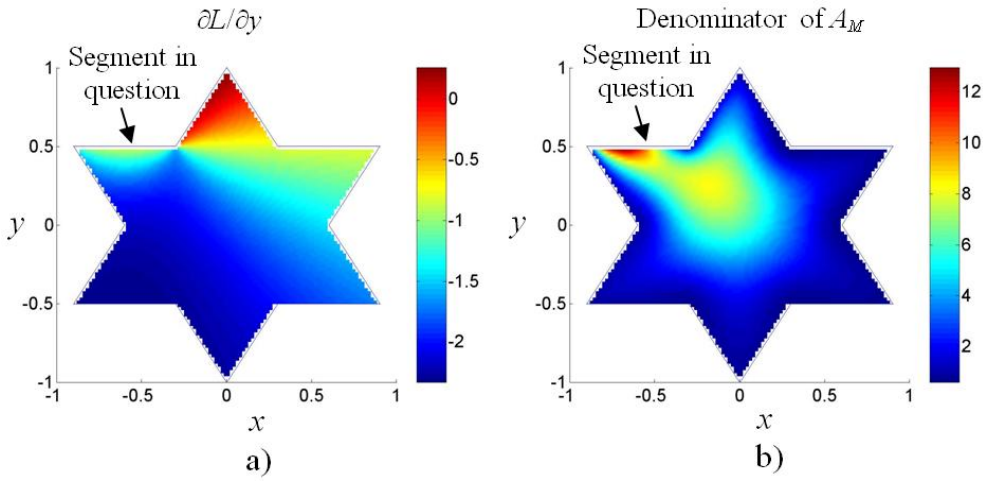


Figure 17. Values of a) $\partial L/\partial y$ and b) (106) for a given segment in a sample concave domain.

7.2 Length factor definition for curved boundary segments

The length factor developed in (105) is only valid for the special case of linear boundary segments. In principle, linear segments are sufficient since any real boundary can be broken into many linear segments until the piece-wise linear boundary arbitrarily approaches the real curved boundary. The TAS in (99) sets no limit on the number of boundary segments, but large numbers of segments do impose some practical difficulties (remember that the exponents on L were reduced to handle larger numbers of segments), including choice of the scaling factor α in (105). While the BVPs explored in this

research have included a tractable number of linear boundary segments (at most twelve), some problem boundaries may be significantly simplified if curved boundary segments can be considered. The three requirements on length factor definition do not restrict the shape of the boundary, nor does the form of the TAS.

The linear case in (105) is extended to arbitrary boundary shapes by expanding on the idea of adding "distances" from the desired interior point to each segment endpoint and subtracting the "distance" between endpoints. Line segment distances are replaced with line integrals along rotated, dilated, and translated copies of the curve defining the boundary segment shape.

Consider an arbitrary relation defining the segment boundary curve C in Figure 18, where the points a and d indicate the ends of the segment. Point e indicates the location within the domain where the length factor is to be calculated. Curve C_1 is generated by dilating and rotating C at point a so that point b_1 on C_1 (which maps to point b on the original C) is coincident with e. The quantity $S_{ae}^{C_1}$ represents the arc length along curve C_1 from points a to e; the arc length $S_{de}^{C_2}$ is defined similarly for a curve C_2 dilated and rotated at point d as in Figure 18. The combination

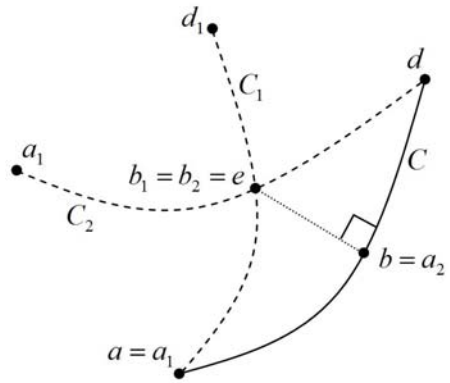


Figure 18. Dilation and rotation of boundary segment curve *C* for length factor definition.

$$S_{ae}^{C_1} + S_{de}^{C_2} - S_{ad}^{C} \tag{110}$$

will satisfy the first length factor requirement of zero boundary value. However, there is no guarantee that

$$S_{ae}^{C_1} + S_{de}^{C_2} > S_{ad}^{C} \tag{111}$$

since the dilation could involve contraction rather than expansion.

Consider another dilation and rotation accompanied by a translation which generates a curve C_3 so that points a_3 and b_3 coincide with points b and e respectively, as in Figure 19. The unsigned distances $S_{ae}^{C_1}$, $S_{be}^{C_3}$ and S_{ab}^{C} are proportional to the lengths of the sides of the dotted line triangle in Figure 19 since dilation is, by definition, a similarity transformation and both rotation and translation do not affect line integrals. The triangle inequality then requires that

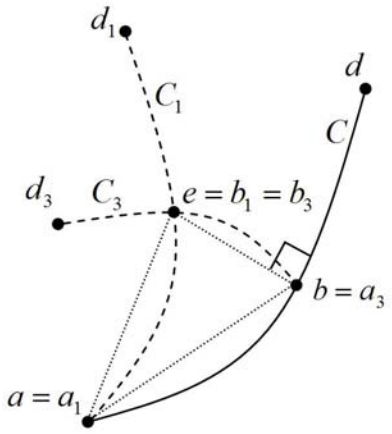


Figure 19. Rotated, dilated, and translated *C* curves and their corresponding triangle.

$$S_{ae}^{C_1} + S_{be}^{C_3} > S_{ab}^{C} \tag{112}$$

A final curve C_4 (not shown) can be generated analogous to C_3 but for point d rather than point a. The combination

$$S_{ae}^{C_1} + S_{be}^{C_3} - S_{ab}^{C} + S_{de}^{C_2} + S_{be}^{C_4} - S_{bd}^{C} = S_{ae}^{C_1} + S_{de}^{C_2} + S_{be}^{C_3} + S_{be}^{C_4} - S_{ad}^{C} \tag{113}$$

then fulfills the first two criteria. The possibility exists that $\partial L/\partial x_j$ for all j could be zero somewhere on the segment when using (113) for L, although this would be a highly unlikely coincidence except possibly for the point equidistance from a and d for a symmetric segment shape with a particular orientation (such as symmetry with a major axis). This coincidental violation of the third criterion at only a single point would affect the solution only if the segment happened to have a Neuman BC. If the segment did indeed carry a Neuman BC, the consequence would be that the BC is not automatically satisfied at that particular point, as is apparent in (90). However, the ANN can adjust to compensate for this discrepancy just as it can at Neuman segment endpoints.

The one drawback with using (113) for a length factor is the need to determine a line perpendicular to C through point e. Multiple points b may exist when more than one perpendicular line intersecting C exists. Consider, for example, the boundary segment defined by the quadratic function $y = x^2$ with segment endpoints $a = (-0.2, 0.04)$ and $d = (1,1)$. Figure 20 illustrates the curves C_1 through C_4 and the candidate perpendicular lines intersecting C for four locations of e along the same horizontal line. The equation to solve for possible b points for a quadratic boundary segment has three roots, two of which are possibly complex conjugates. Different locations for e produce either a single choice for b, or three possibilities. The real point b closest to e is chosen in the case of multiple possibilities. The chosen b jumps discontinuously from one side to another when e crosses from the second to the first quadrant, as in Figure 20(a) and (b). Two of the perpendicular lines suddenly disappear (the roots become complex conjugates), when e is moved farther, as in passing from Figure 20(c) to Figure 20(d). No strategy

could be found for choosing b to avoid discontinuities, for even a simple quadratic segment shape.

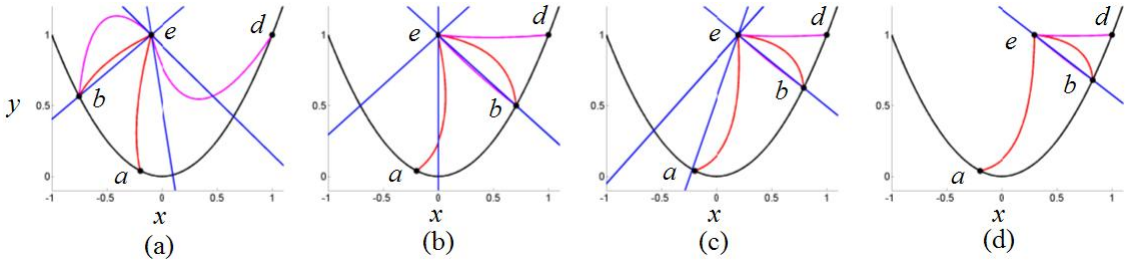


Figure 20. Basis for length factor on a quadratic-shaped segment at various points *e*.

Consider the length factor

$$L = \alpha\left[\left(c_1 + \alpha_3 c_3\right) S_{ab}^{C} + \left(c_2 + \alpha_4 c_4\right) S_{bd}^{C} - S_{ad}^{C}\right] \tag{114}$$

where the dilation factors c_i originate from the amount of dilation in determining the curve C_i, e.g. $S_{ae}^{C_1} = c_1 S_{ab}^{C}$. The scaling factor α plays the same role as in (105) and the factors α_3 and α_4 will be considered later. Notice that (113) and (114) are equivalent when $\alpha = \alpha_3 = \alpha_4 = 1$. The discontinuities along the vertical line $x = 0$ foreshadowed in Figure 20 are evident in Figure 21, which displays contours of constant length factor using (114) with unity scaling factors. The factors α_3 and α_4 are introduced into (114) in order to lessen and/or remove the impact of any discontinuities. Adding these factors does not affect the first requirement on length factors since $c_3 = c_4 = 0$ whenever e falls on C and so L will return zero. As long as α_3 and α_4 are greater than unity, the inequality in (112) will not be affected. The length factor contours in Figure 22 are without significant discontinuities in value by using values of $\alpha = \frac{1}{2}$, $\alpha_3 = 10$, and $\alpha_4 = 1$.

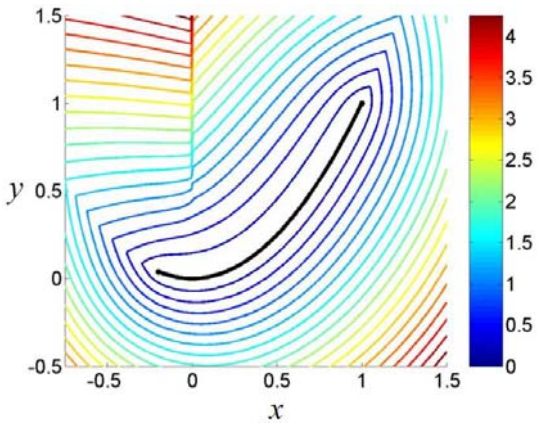


Figure 21. Length factor with discontinuities in value for a quadratic segment.

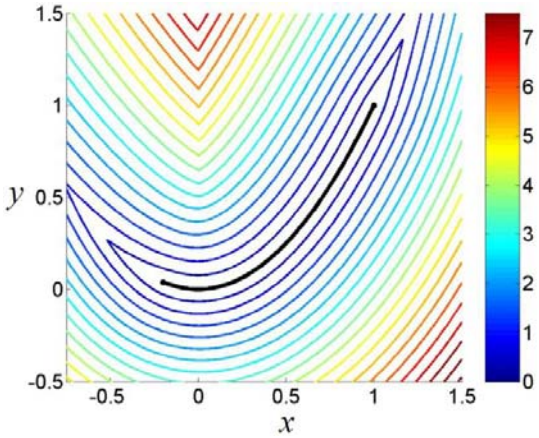


Figure 22. Length factor without discontinuities in value for quadratic segment.

Parameters α_3 and α_4 chosen by trial and error may not sufficiently remove discontinuities for segments of all shapes. Exploration of other techniques for length factor definition may be required if the current method proves unsatisfactory in future numerical experiments. For example, the choice of b as lying on a line perpendicular to C is not a requirement; the location of b must simply be a function of e, and the two points b and e must coincide whenever e falls on C somewhere between a and d. If another algorithm for locating a valid b proves difficult, b could possibly be fixed at some point on C and the length factor definition adjusted to handle the stationary point.

These possibilities have not yet been explored since the length factor for linear boundary segments in (105) has been sufficient for all computations undertaken to date.

7.3 Comparison of three different length factor definitions

The length factors in (105) and (114) for linear and curved boundary segments, respectively, were developed to serve as a standard definition which could be used for any boundary. Simpler and/or more effective length factor definitions may be available for certain boundary segment shapes. Consider, for example, a circular boundary. Considering the circle as a single boundary with a length factor involving radii is certainly more desirable than breaking the boundary into numerous linear segments. Equations (105) and (114) provide a standard length factor definition, but the user should be aware that some boundary geometries may lead to more convenient length factor definitions.

7.3.1 Problem considered

Three different length factors are now considered for solving the Laplace equation on the domain in Figure 23 composed of a circular boundary and a square hole. Both the circle and square hole in the center have Dirichlet conditions. Solving the Laplace equation on this domain is equivalent to a heat transfer problem where the hole acts as a heat sink with its zero BC. The length factor

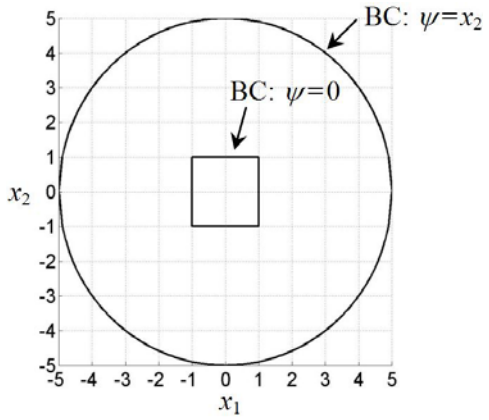


Figure 23. Domain and BCs used to compare length factor definitions.

$$L_{\text{circle}} = r - \sqrt{x_1^2 + x_2^2} \tag{115}$$

for the circle of radius $r = 5$ satisfies all length factor criteria with a singularity only at the circle center (the singularity is in the second partial derivatives), a point which is not within the problem domain.

A valid length factor for the square hole is not as naturally defined. Perpendicular distance from each of the four sides cannot be used since lines defining the sides extend into the domain (the light red lines in Figure 24). The standard length factor from (105) could be used, however, once for each side.

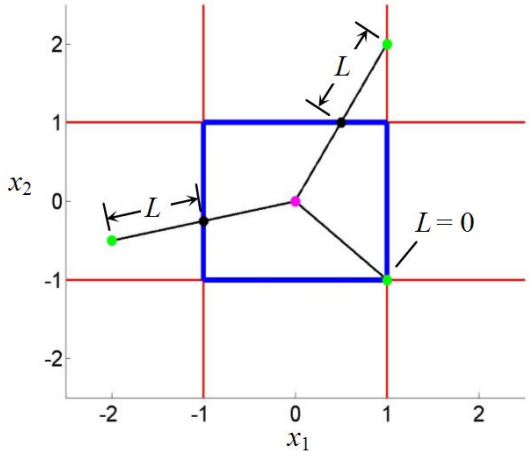


Figure 24. Illustration of a length factor considering the square hole as one segment.

It would be convenient if the hole could be considered as a single segment. Consider a line drawn from the desired (x_1,x_2) point to the center of the square hole. The length factor can then be defined as the distance from (x_1,x_2) to the intersection of the line and the square boundary, as illustrated by three examples in Figure 24. Values of the length factor L_{basic} defined in this fashion appear in Figure 25, including contour lines of uniform length factor value projected onto the x_1,x_2-plane. This length factor contains obvious discontinuities in slope along lines aligned with the corners of the square hole. These discontinuities arise from the abrupt shift in the side of the square first intersected when (x_1,x_2) crosses the diagonal lines passing through opposite corners.

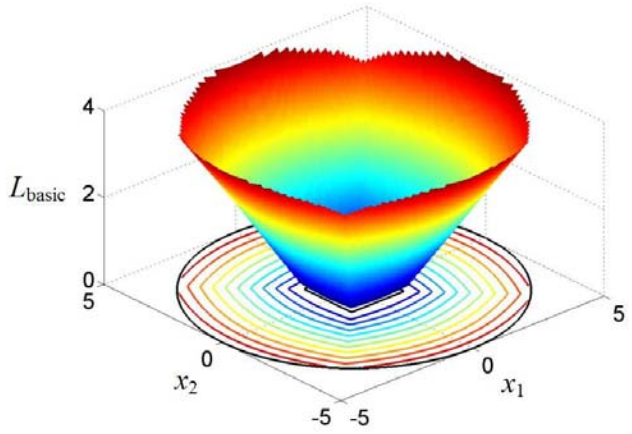


Figure 25. Plot of the basic length factor for the square hole.

The protracted struggles with discontinuities and singularities in development of the TAS, documented earlier in this dissertation, prompted consideration of an alternative length factor for a single segment representing the square hole. The fact that the hole sides correspond to horizontal and vertical lines at values of positive and negative unity was exploited by considering the complex mapping

$$z = \left(x_1 + x_2 i\right)^4 \tag{116}$$

The lines comprising the four square sides are coincident in this mapping and all four square corners collapse to a single point as shown in Figure 26. The square is teardrop-shaped in the mapping where the coordinate axis origin falls inside the teardrop. A new length factor L_{map} is defined, again determined by drawing a line from the desired point to the hole center. This time however, the distance to the intersection of the hole is measured in the mapping space where all four sides are coincident. The same three sample points from Figure 24 appear in Figure 26 as examples of length factor computation. The slope discontinuities apparent in L_{basic} are absent in L_{map} as is evident in Figure 27. The only remaining discontinuities derive from the point on the teardrop which corresponds to the corners of the square. These discontinuities are imperceptible in Figure 27, and would not be expected to affect training.

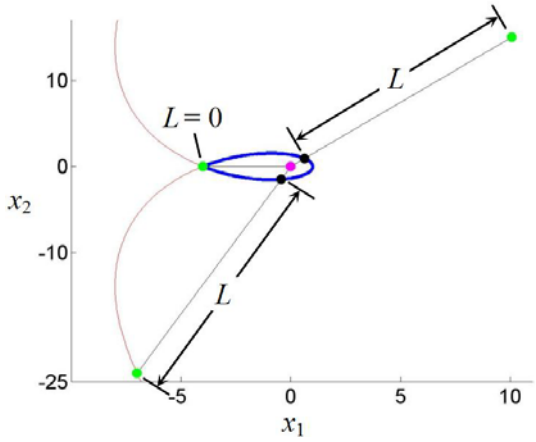


Figure 26. Illustration of a length factor for the square hole using a complex mapping.

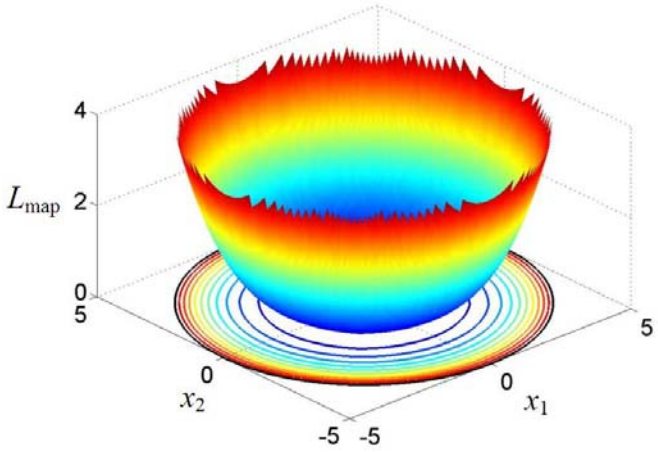


Figure 27. Plot of the length factor for the square hole computed using a complex mapping.

The third length factor for comparison uses the standard length factor definition in (105) with $\alpha = \frac{1}{2}$ for each of the four segments composing the square hole. The composite length factor produced by multiplying the length factors of each of the four hole segments appears in Figure 28. Note that the length factor shape in Figure 28 is very similar to that in Figure 27 although the magnitude is significantly larger. Remember that L_{comp} in Figure 28 is calculated by multiplying four length factors together; the maximum value of each individual length factor is approximately 5, comparable to the maximum values of L_{basic} and L_{map}. The magnitude of L_{comp} can be adjusted by varying the value of α. Performance is insensitive to α, with essentially the same error norms for values $0.2 \le \alpha \le 1.2$ (as long as appropriate training points are chosen for T).

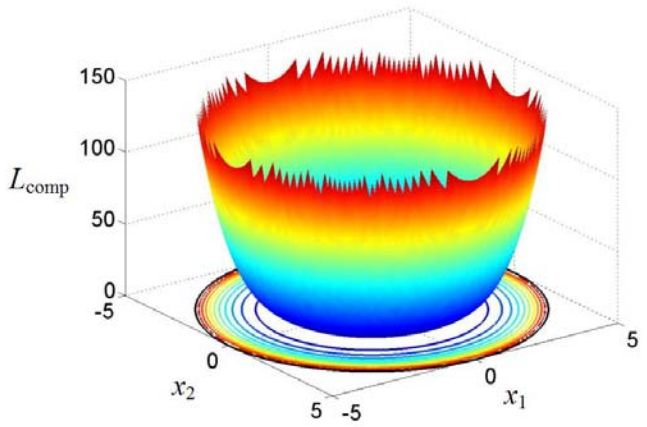


Figure 28. Plot of the composite length factor for the square hole using the standard length factor definition.

7.3.2 Results and comparison

The perpendicular distance length factor in (115) is used for the circular boundary segment and results for the three length factors L_{basic}, L_{map}, and the standard length factor definition (105) generating L_{comp} are compared when solving the Laplace equation with the BCs illustrated in Figure 23. This problem was first investigated before development of (105), and so L_{basic} was first conceived to properly define a length factor. The slope discontinuities in Figure 25 led to the complex mapping producing the discontinuity free L_{map}. The eventual development of (105) provided a third possible length factor for comparison.

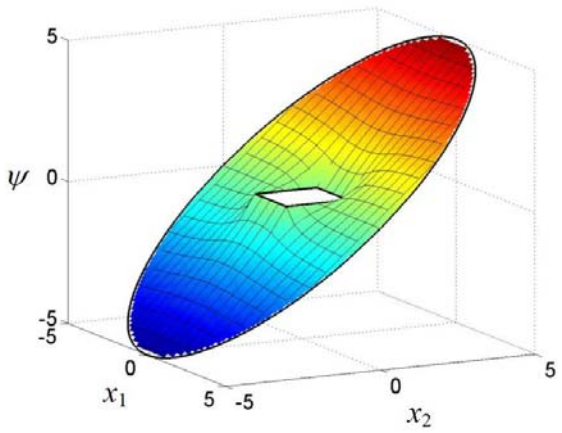


Figure 29. FEMLAB® solution to the Laplace equation on a circular domain with a square hole.

The "true" solution to this problem appears in Figure 29 as calculated using FEMLAB®. Approximate solutions were trained for each of the three length factors for the square hole using 15 hidden nodes and a set T of training points determined by all points on an evenly spaced 12×12 grid falling within the domain. Table 1 lists error norms when evaluating (57) on a 100×100 grid for the three choices of length factor for the square hole. Length factors L_{map} and L_{comp} produced nearly identical solutions as expected from the similarities between Figure 27 and Figure 28. Surprising, however, is the superior performance of L_{basic}, which contains discontinuities in slope aligned with the hole corners. The existence of discontinuities in the length factor in fact aided determination of the approximate solution since the true solution also contains slope discontinuities near the hole corners. This comparison illustrates how the user can infuse knowledge into the system by designing a length factor with characteristics known to appear in the solution. In this manner, the educated user can transfer knowledge to the "black-box" ANN through judicious length factor definition.

Table 1. Comparison of performance with three length factor definitions for the square hole.

Length factor for square hole	Error norm
L_{basic}	0.052
L_{map}	0.24
L_{comp}	0.26

The problem considered in this section was originally chosen not to compare length factors, but to compare with results from [20] when solving the same problem. The method used in [20] did not automatically satisfy BCs and subsequently the error function in (21) was employed. Results were reported in [20] at four selected points within the domain and five others on the domain boundary. Error norms calculated using only these points are calculated to be 0.13 and 0.059 for points within the domain and on the boundary, respectively. Corresponding results from Table 1 using L_{basic} are significantly better at 0.052 and 0.000 (error norm evaluated anywhere on the domain boundary will be exactly zero by definition). While the nine points are the only numerical results appearing in [20], a plot of the generated TAS is provided. Comparison of the plot from [20] with Figure 29 reveals that the former is highly inaccurate, especially near the center hole. The slope discontinuities apparent in Figure 29 are completely absent from the plot in [20] (which is also the case when using L_{map} and L_{comp}). The four chosen points within the domain in [20] are relatively close to the circular boundary, precisely the region of the domain which is most accurate judging from the plot (the error norms appearing in Table 1 include over 7400 evenly spaced points).

The TAS from [20] solving this problem is relatively flat for most of the domain interior with values of $\psi_t \approx 0$. This result is important since it exemplifies a significant

pitfall with using ANNs to solve BVPs. A constant TAS of $\psi_t = 0$ exactly satisfies the Laplace equation everywhere in the domain as well as the square hole BC, incorrect only on the circular domain boundary. Similar results were reproduced using the method here by omitting training points close to the circular boundary from *T*. In such a case, error in the DE quantified by (58) was quickly reduced to near zero at all training points, giving false indication that an accurate solution had been obtained. The ANN is, of course, still capable of correctly representing the true solution even with training points restricted in this manner, but takes the "path of least resistance" during training toward the local minimum with solution $\psi_t = 0$. While ANNs are a powerful and relatively simple tool for solving BVPs, care must still be taken in choosing design parameters and in interpreting the results.

8 Further computational results

Four example problems are solved using the approach developed for two-dimensional BVPs. The first two problems solve the Laplace equation on a non-symmetrical piece-wise linear domain first with only Dirichlet conditions, and then with a mixture of Dirichlet and Neuman BCs. The Dirichlet example has a particularly complex solution due to discontinuous BCs at segment intersections. The second two problems solve a nonlinear partial differential equation on a more complicated star-shaped domain, again first with all Dirichlet and then with mixed BCs. These second two examples illustrate how the solution with the approach here compares with another ANN method for solving BVPs on irregular boundaries from the literature [21]. The performance of the following examples is evaluated using the error norm in (57), where ψ is either the exact analytical solution or the results from a reliable commercial application such as FEMLAB®. The error norm will be calculated for the same set T as was used for training as well as another set T based on a much finer 100×100 square grid.

8.1 The Laplace equation on a polygon-shaped domain

The solution to the Laplace equation inside a five-segment piece-wise linear domain appears in Figure 30(a) where the chosen BCs require constant values for the solution on each segment. The solution was obtained with FEMLAB®, and is used as the "analytical" solution ψ in the error norm equation (57). Note that the BCs specify a discontinuous value for ψ at four of the five segment intersections.

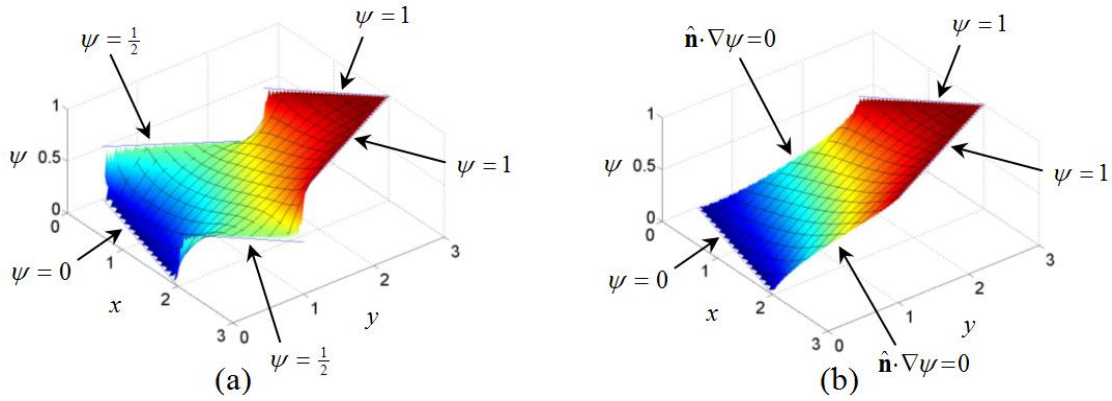


Figure 30. Laplace equation solutions with (a) Dirichlet and (b) mixed conditions (polygon-shaped domain).

The second problem investigated uses the same domain boundary as the first, but replaces two of the Dirichlet boundaries with Neuman boundaries as in Figure 30(b). Again, a solution obtained from FEMLAB is used as the exact "analytical" solution ψ. Remember that the functions $\hat{\mathbf{n}}_i(\mathbf{x})$ and $g_i(\mathbf{x})$ associated with segment i must return specific values for $\mathbf{x}$ on segment i without any requirement on their values for $\mathbf{x}$ not on the given segment. BCs for these two problems specify uniform values and so $g_i(\mathbf{x})$ can simply assume constant values. The normal vector for a linear boundary segment is also uniform, resulting in

$$\hat{\mathbf{n}}_i(\mathbf{x}) = n_{i1}\hat{\mathbf{i}} + n_{i2}\hat{\mathbf{j}} = \pm\frac{m}{\sqrt{m^2+1}}\hat{\mathbf{i}} \pm \frac{1}{\sqrt{m^2+1}}\hat{\mathbf{j}} \tag{117}$$

where m is the slope of the boundary segment. The ± sign is chosen to ensure an inwardly-directed normal vector, and is determined by the sign of the slope and the orientation of the line with the boundary.

8.2 A nonlinear DE on a star-shaped domain

The second two problems involve solving the nonlinear, non-homogenous equation

$$\nabla^2\psi\left(x_1,x_2\right)+e^{\psi\left(x_1,x_2\right)}=1+x_1^2+x_2^2+\frac{4}{\left(1+x_1^2+x_2^2\right)^2} \tag{118}$$

considered by the authors of [21, 27] who introduced automatic satisfaction of BCs through judicious definition of the TAS. The DE in (118) is solved here to compare with results from [21] where the authors settled for satisfying BCs on irregular domains at discrete points rather than everywhere along the boundary.

Verifying that the equation

$$\psi\left(x_1,x_2\right)=\ln\left(1+x_1^2+x_2^2\right) \tag{119}$$

satisfies (118) is straightforward. Both Dirichlet and Neuman BCs can then be artificially created consistent with the solution in (119) for whatever domain shape is desired. Defining the problem in this manner produces a known analytical solution to a non-homogenous and nonlinear DE for any desired domain shape. The domain boundary is defined by a six-pointed star, and one example problem solves (118) with all Dirichlet conditions while another considers only one segment with a Dirichlet condition. The values of (119) within this star-shaped domain appear in Figure 31.

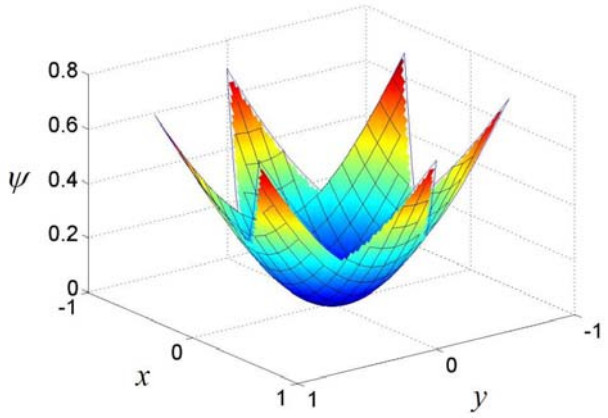


Figure 31. Solution to the nonlinear example problem (star-shaped domain).

Dirichlet BCs are determined by evaluating (119) at the specified boundary. Clearly, equating $g_i(\mathbf{x})$ to (119) would provide the TAS with the exact analytical solution and training would quickly force $A_M \approx F \approx 0$ to result in a near perfect solution. This also emphasizes how the user can accelerate training by incorporating any a priori knowledge of the true solution into the functions g_i (or into the length factors as presented in the length factor comparison presented earlier).

The boundary function

$$g_i\left(x_1, x_2\right) = \ln\left[1 + x_1^2 + \left(m_i x_1 + b_i\right)^2\right] \tag{120}$$

is chosen instead where m_i and b_i are the i^{th} boundary segment slope and intercept, respectively. This definition of $g_i(\mathbf{x})$ is the correct solution only on boundary segments, and not at other locations in the domain. Equation (120) is coincidentally only a function of x_1 and not of x_2, emphasizing the fact that the user has complete freedom to set values of g_i within the domain (the only requirement is on values of g_i for points on segment i). The function for Neuman BCs is determined by first computing the partial derivatives

$$\frac{\partial \psi}{\partial x_1} = \frac{2x_1}{1 + x_1^2 + x_2^2} \text{ and } \frac{\partial \psi}{\partial x_2} = \frac{2x_2}{1 + x_1^2 + x_2^2} \tag{121}$$

of (119) and combining them with $\hat{\mathbf{n}}_i(\mathbf{x})\cdot\nabla\psi(\mathbf{x}) = g_i(\mathbf{x})$ and (117) to produce

$$g_i(x_1,x_2) = \pm\frac{2x_1}{1+x_1^2+x_2^2}\frac{m_i}{\sqrt{m_i^2+1}} \pm \frac{2x_2}{1+x_1^2+x_2^2}\frac{1}{\sqrt{m_i^2+1}} = \frac{2(\pm m_i x_1 \pm x_2)}{(1+x_1^2+x_2^2)\sqrt{m_i^2+1}} \quad (122)$$

8.3 First-order solutions using A_D

It is interesting to explore the effect on the TAS by satisfying the Dirichlet BCs before training the ANN to solve these problems. The TAS has no error on any Dirichlet boundary segments, and presumably the error would be small near the boundaries. All the information represented by the Dirichlet BCs is coded into the A_D term of (99). The length factors, which represent a distance from each boundary segment, are used to ensure that A_D returns the correct value on each Dirichlet boundary segment. When inside the domain, the value of A_D is essentially an average value of the Dirichlet BCs weighted by the local length factor values. The result is a function A_D surprisingly close to the true solution even before ANN training begins.

Figure 32(a) shows the function A_D for the polygon-shaped problem with only Dirichlet BCs, and Figure 32(b) depicts the residual error of A_D when compared with the true solution. The shading of the residual error in Figure 32(b) is adjusted so that black corresponds to zero error with lighter shades representing errors with larger absolute values. Figure 32(a) is nearly indistinguishable from the true solution in Figure 30(a) and boasts an astonishingly low error norm of 0.068 when evaluated on a 100×100 grid. The length factors, though not specifically designed for this purpose, provide an accurate basis for averaging A_D within the domain. Starting with such an accurate approximate solution is a major reason why this ANN method is simpler and yet often more accurate than other ANN methods for solving BVPs. The function A_D provides a solid basis and

the ANN need only fine tune the TAS to achieve even better accuracy. The job of the ANN has been significantly simplified by infusing the TAS with a priori knowledge about the solution (via Dirichlet BCs) and the problem geometry (via length factors). Length factor development is a clever approach for automatically satisfying BCs with the added benefit that they help provide knowledge about the solution value even inside the domain.

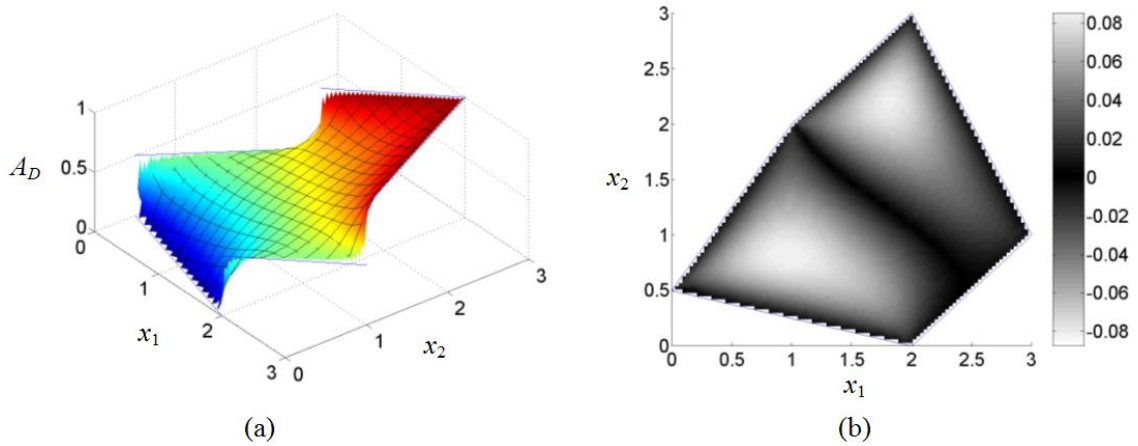


Figure 32. Plot of a) A_D and b) its residual error for the polygon problem with Dirichlet BCs.

Figure 33(a) and (b) display the value and residual error, respectively, of A_D for the polygon-shaped problem with mixed BCs. The error norm of 0.15 for this case is also quite low despite having only three of the five boundary segments with Dirichlet conditions. Residual errors are largest near Neuman boundaries and at the interior of the domain. Still, the main features of Figure 33(a) are directly comparable to the true solution in Figure 30(b).

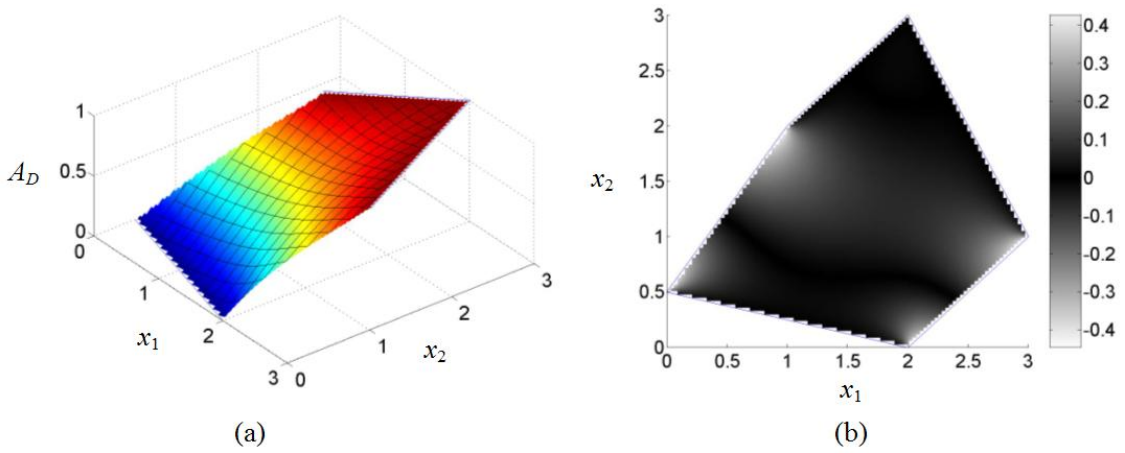


Figure 33. Plot of a) A_D and b) its residual error for the polygon problem with mixed BCs.

The values and residual errors for the star-shaped problem with all Dirichlet and with mixed BCs appear in Figure 34 and Figure 35, respectively. The value of A_D in these cases is not as close to the true solution in Figure 31; errors norms are 0.88 for the all Dirichlet case and 2.4 for the mixed BC case. The true solution is zero at the center of the domain, a value impossible to obtain from an average of the Dirichlet BCs. The DE (118) solved in these cases is nonlinear and nonhomogenous, which produces values inside the domain beyond the range of the Dirichlet BCs. Even so, A_D in Figure 34(a) provides a good starting point for training the ANN.

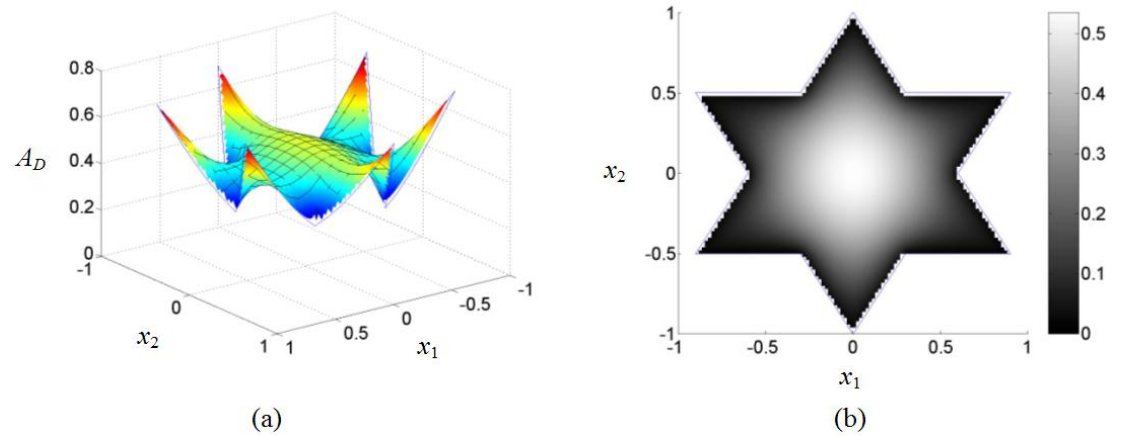


Figure 34. Plot of A_D and its residual error for the star problem with Dirichlet BCs.

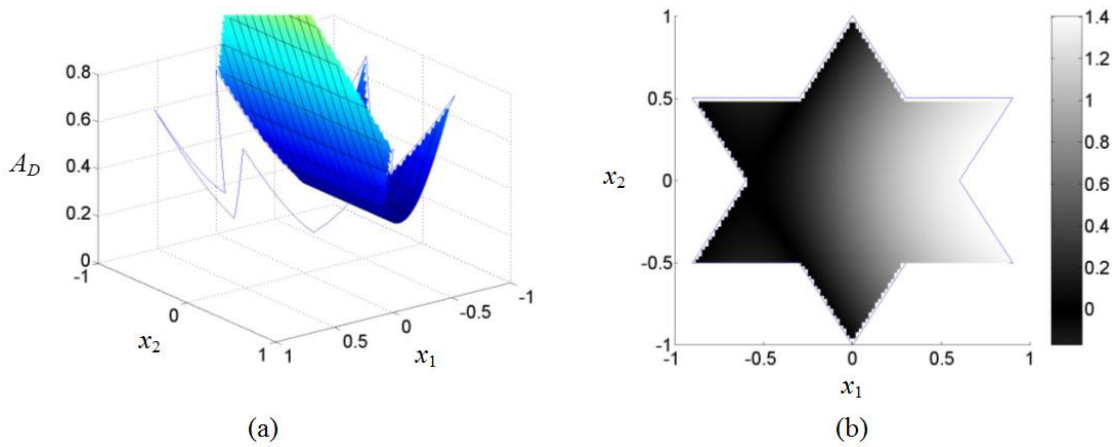


Figure 35. Plot of a) A_D and b) its residual error for the star problem with mixed BCs.

Residual errors for the mixed case in Figure 35 are significantly worse due to the single segment with Dirichlet BCs (although a second segment has coincidentally correct values due to domain symmetry and (120) being only a function of x_1). In any case, A_D in Figure 35 is still a better starting point than having no knowledge whatsoever of the true solution, as is the case with most other ANN methods.

In general, residual error in A_D will be lower for domains with many Dirichlet segments, and for simpler DEs like the homogenous Laplace equation considered in Figure 32 and Figure 33. Low residual error in A_D may or may not translate into an improved TAS after training the ANN; this depends primarily on the complexity of the problem to be solved and the choice of design parameters. However, training time is significantly reduced for problems with more accurate A_D values. This was especially apparent when comparing the training times of solving the problem in Figure 34 as opposed to the one in Figure 35.

8.4 Choice of design parameters

As in most ANN applications, the number of nodes in the hidden layer, H, has a direct effect on the quality of the solution. However, fine tuning of H has not been

observed to be necessary; performance is not exceedingly sensitive to small changes in H, as indicated previously in Figure 7. ANNs are first trained with a relatively small value for H which is later increased if the error in the DE, that is (5) using (58), is not reduced to acceptable levels. Large values for H are avoided since they significantly increase computation time.

The other design parameter is the number and location of points to include in T. Sufficiently many points spread throughout the domain are necessary in order to avoid overfitting and to produce a result with good generalization. Too many points in T, however, can slow training significantly. The simplest method for deciding T is to choose all points on a uniform $n \times n$ grid which fall within the domain. A larger value for n is required whenever training produces G values significantly larger at locations between the points in T, i.e. when overfitting is experienced early on during training. Remember that the TAS is continuous everywhere in the domain, and the error in the DE measured by G is readily available at any point in the domain.

A value for n required for good generalization may include points close to boundary intersections where singularities can undermine proper training. A point in T with an unreasonably large G can receive a disproportionate amount of attention during training and hinder development of a good solution over the entire domain. One way to minimize this effect is to scale the length factor with a value α less than unity. However, very small values for α reduce the ANNs effectiveness by making the A_M and F terms in (99) very small (these terms involve the ANN output N), effectively crippling the ANN's influence over the solution.

The effect of singularities at boundary intersections can also be suppressed by simply removing points close to them from the training points in T. Manual determination of which points are detrimental to training requires careful and time-consuming observation, which necessitates development of an automatic algorithm for determining which points to include in T. Difficulties are only encountered at points close to the boundaries and so a threshold on the minimum length factor is imposed. The absolute value of G must be below a second threshold if a point "close" to a boundary is to be considered. In this method, all points on the interior of the domain are automatically included, and points closer to the domain boundary are added if their values of G are sufficiently small to avoid interfering with development of an accurate TAS. This method is especially effective for boundaries with many Dirichlet boundaries since the TAS is already exact there. The algorithm for determining membership in T is summarized as follows:

1. Consider all points within the domain lying on an $n \times n$ grid
2. Include a point for which $\min\left(L_1, L_2, \ldots, L_{D+M}\right) < \beta$ only if $|G| < \gamma$

8.5 Results for the example problems

Table 2 summarizes the design parameters used for each of the four problems as well as error norm values before and after training. Roughly the same parameters are used for all four example problems. Dependence on the value of α, which scales the length factor in (105), is stronger than for the other parameters since it essentially regulates the amount of influence the ANN has over the TAS in (99). The star-shaped domain from Figure 31 includes twelve boundary segments (as opposed to the five segments from the polygon-shaped domain in Figure 30) and thus requires a value for α

closer to unity since successive products of small length factors will reduce the magnitude of the F term in (99) which includes the ANN output N. The other design parameters required little adjustment to achieve an acceptable error norm.

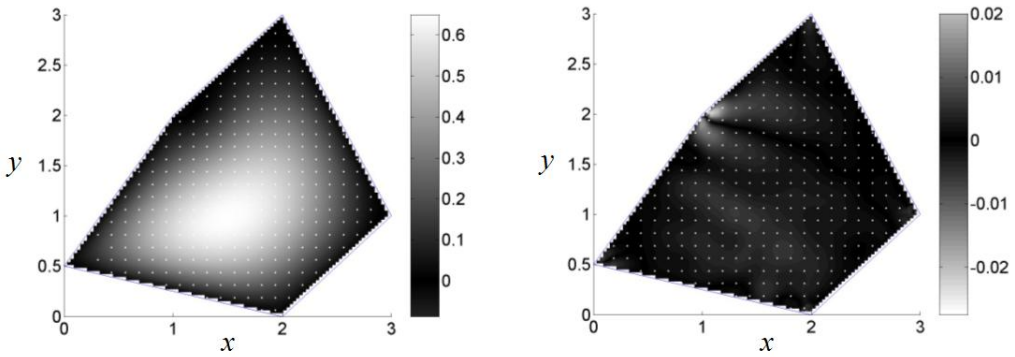
Table 2. Design parameters for the example problems with performance before and after training.

Design parameter	Polygon problem		Star problem	
α	0.5		0.8	
β	5		5	
γ	0.15		0.15	
n	24		28	
H	40		45	
Error norm	Dirichlet	Mixed	Dirichlet	Mixed
Before (100×100)	0.52	0.57	0.32	1.9
After (100×100)	0.0057	0.039	0.078	0.069
After (n×n)	0.0058	0.039	0.085	0.071

Three values for the error norm are presented in Table 1 for each of the four example problems. The first includes all the points on a 100×100 grid for the set T and is evaluated before training commences with random ANN weights. The error norm is already reasonably low despite random ANN weights since Dirichlet BCs require an exact solution on such boundary segments. The errors norms before training are not as low as those reported previously for A_D alone since including the A_M and F terms dependent on the ANN adds a contribution due to the random starting weights. The untrained TAS still provides a reasonably accurate solution as a starting point for training. This effect is especially significant for domains with primarily Dirichlet segments and for small values of α. Choosing α to be small effectively scales back the F contribution from the ANN, leaving a TAS composed essentially of only A_D. Values too small, however, require exceedingly large ANN weights to compensate and strain the ability of the ANN to adjust itself appropriately.

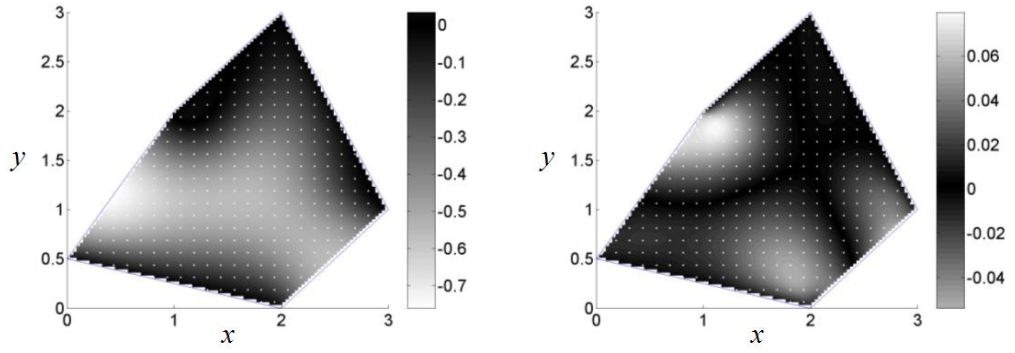
The other two error norm values in Table 1 indicate performance after the ANN is trained. Training is halted when DE error in (5) using (58) begins to rise when considering points in T on a 100×100 grid rather than the $n \times n$ grid used to update ANN weights. Once training is completed, error norm is evaluated on both the 100×100 and $n \times n$ grids in order to gauge the solution's ability to interpolate between training points. Performance is roughly equivalent on the significantly finer 100×100 grid. In some cases, especially for problems with many Dirichlet boundary segments, performance is better between training points since points closer to the boundary have larger representation when the finer grid can match the irregular boundary more closely. These results show that a properly configured ANN, that is, one with well-chosen design parameters, produces a solution whose error everywhere between training points is on the same order as at the points themselves.

The data in Table 2 represent average performance over the entire domain. Residual error in the solution as a function of location, from (2), is depicted for the polygon-shaped domain with Dirichlet and mixed BCs in Figure 36 and Figure 37, respectively, and for the star-shaped domain with Dirichlet and mixed BCs in Figure 38 and Figure 39, respectively. The shading in Figure 36 through Figure 39 is scaled so that dark corresponds to zero error and light to the maximum magnitude of absolute error. The light dots in these figures indicate locations in the domain included in the set T of training points.



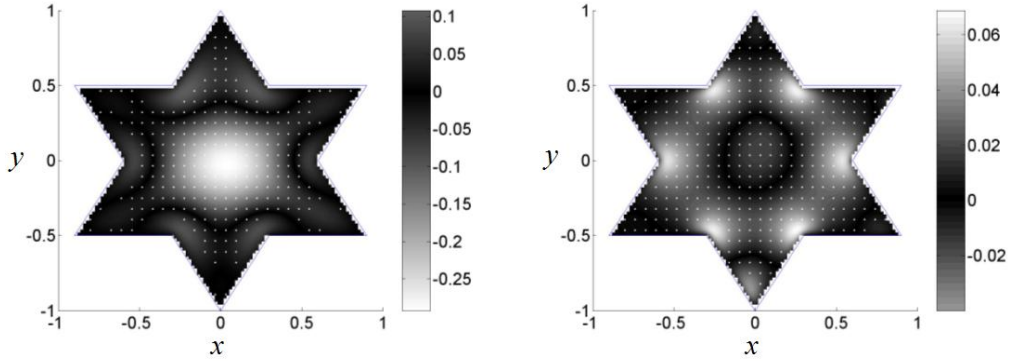
(a) (b)

Figure 36. Residual solution error (a) before and (b) after training for Dirichlet BCs on the polygon domain.



(a) (b)

Figure 37. Residual solution error (a) before and (b) after training for mixed BCs on the polygon domain.



(a) (b)

Figure 38. Residual solution error (a) before and (b) after training for Dirichlet BCs on the star domain.

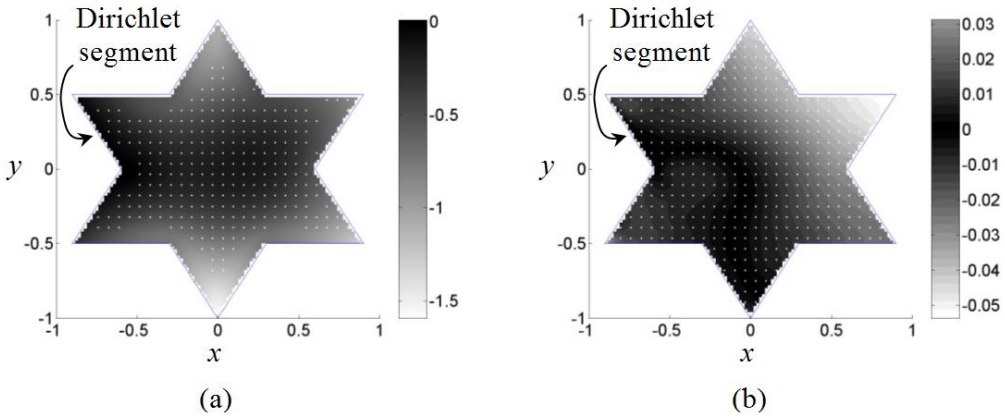


Figure 39. Residual solution error (a) before and (b) after training for mixed BCs on the star domain.

Essentially all points falling on the $n \times n$ grid are included in the set of training points by the time training ends, for example see Figure 38(b). Those excluded when training commences due to large values of G, such as in Figure 38(a), are eventually added by an included neighboring point reducing the magnitude of G in the surrounding area and allowing other points to move below the γ threshold. Systematically adding points to the set T during training greatly increases performance of the fully trained ANN as opposed to including all points in the beginning; points with excessively large G values will receive disproportionate attention during gradient descent update and hinder convergence on a solution with good generalization. By the end of training, regions important to a satisfactory solution will be represented as long as sufficiently many points are included from the interior of the domain, that is for a sufficiently small β threshold. Final performance is strongly dependent on the choice of the β threshold only in the case where the domain consists primarily of Neuman boundary segments, such as in Figure 39. Inclusion of points close to the boundary is not critical for Dirichlet segments since the solution at the boundary is already exact. The β threshold must be

chosen carefully for domains with mainly Neuman segments; thresholds too small will skew results by including training points with large G values, and thresholds too large will allow the solution to become irreparably poor near the boundary before points in that region can be added to the training set T.

The solution error D is not uniform over the entire domain, as illustrated by Figure 38 and Figure 39. Regions of relatively high error generally originate from segment intersections where singularities in the second derivative of the length factor can hinder training. Removal of singularities and/or suppression of their effects have been a major emphasis in the research leading to the development of the TAS and length factor definition. Results presented here are acceptably accurate for most engineering applications with error norms on the order of 10^{-2}. Continued research should produce methods for reducing the effect of singularities, thereby bringing the error norm down even further.

Gradient descent techniques for ANNs do not produce symmetrical solutions even when the true solution is; compare, for example, solutions appearing in Figure 38 and Figure 39 with the true solution (119) appearing in Figure 31. Even so, structure imposed by the influence of length factors, especially for Figure 38 with all Dirichlet boundary segments, does produce a nearly symmetrical solution. Post processing the final TAS to impose a symmetrical solution would likely produce a more accurate solution. And as detailed earlier, problems with many Dirichlet boundary segments benefit from exact boundary values, this time with a greater influence over symmetry of the solution.

Reference [21] also solved the example problems on the star-shaped domain. Numerical results were not included in [21] although a plot of residual errors in the TAS was provided for the problem with Dirichlet BCs. From this plot, an error norm of approximately 1×10^{-5} is estimated. Although the method used in [21] reports superior performance, it is significantly more complicated and adjusts ANN weights with an unconventional optimization technique [16] rather than the familiar Levenberg-Marquardt technique used here. Besides its simplicity, the method presented here satisfies arbitrary BCs everywhere on irregular domains as opposed to only at specified points on the boundary as in [21].

9 An essential theorem for solution convergence

Any function can be approximated to an arbitrary level of accuracy with some combination of ANN weights [14]. Recall that the simplest ANN method for solving BVPs in (46) uses the ANN output directly as the approximate solution. It is clear in this case that the ANN can represent the true solution ψ to an arbitrarily small error. Not as clear is the case of the TAS in (99) where A_M and F are functions of the ANN output N. Consider the special case where $A_M = 0$ (that is where all boundary segments have Dirichlet conditions). The universal approximator ANN output N would then adjust itself to represent a different function, namely

$$N \approx \frac{\psi - A_D}{\prod_{j=1}^{D} L_j} \tag{123}$$

The function to be represented for the general case with mixed BCs cannot be solved algebraically since A_M involves partial derivatives of N. However, the solution of the resulting partial differential equation for N would be the requisite function to be represented by the ANN.

The universal approximator property of ANNs assures that some combination for the ANN weights exists for which the difference between the TAS ψ_t and the true solution ψ is arbitrarily small. The question remains, however, if an ANN trained to reduce the DE error can adjust ANN weights to values acceptably close to the optimal ones.

The natural criticism of "black-box" ANN methods for solving BVPs is the lack of an error bound on the TAS. While the gradient descent training algorithms used in this work are proven to be fast and effective at reducing error and producing a good solution, they do not provide any error bounds on the solution. At first glance, the application of more modern training algorithms with error bounds such as those employed by support vector machines [31] would solve this problem. However, such methods bound the error function used to update model parameters. The result would be a bound on the error in the DE, which has no obvious translation into a bound in the error of the solution itself.

The theorem presented in this chapter links reductions in the DE error to reductions in the error of the solution itself, and may be stated as follows: *the difference between the true and approximate solutions will decrease everywhere in the domain when the error in the DE is also reduced everywhere in the domain*. In practice, error in the DE cannot be expected to reduce at *all* locations in the domain simultaneously, especially those influenced by singularities at segment intersections. Numerous numerical experiments have shown, however, that error in the DE will reduce for essentially the entire domain when the ANN is well-designed.

Imagine a TAS which coincidentally produces the correct solution at a certain $\mathbf{x}$ but has large errors near $\mathbf{x}$. While error in the solution will be zero at this point, error in the DE will certainly be large since the *shape* of the TAS at that point is incorrect even though the *value* at that point is exact. An analogous scenario is possible where the shape of the TAS is correct (that is the DE is satisfied) and yet the value of the TAS is incorrect. Consequently, low error in the DE at any given point in the domain is a

necessary but not sufficient condition that the solution error will also be low there. Consider now that the shape of the TAS is consistent with the DE *everywhere* in the domain. The TAS must certainly be a correct particular solution to the DE. It could, however, be a particular solution inconsistent with the required BCs. But since the BCs are already satisfied exactly, a TAS satisfying the DE everywhere in the domain *must* be the particular solution sought.

The intuitive argument from the previous paragraph suggests that improved global satisfaction of the DE should lead to a commensurately more accurate TAS. The convergence theorem presented here begins by assuming that the DE is reduced everywhere in the domain and uses the fact that automatic satisfaction of Dirichlet BCs "anchors" the TAS at the correct values. This anchoring ensures that the correct particular solution to the DE is approximated by the TAS, which becomes closer to the true solution as error in the DE is reduced in successive training iterations.

Numerous numerical experiments have substantiated the intuitive argument by establishing a strong correlation between DE error and solution error. Computational results also suggested the requirement that DE error must be reduced everywhere in the domain. Choosing too few points for the training set results in overfitting where DE error is consistently reduced at the training points, but worsens at locations between, leading rapidly to an increasingly inaccurate TAS. Observing fitness of the DE between training points during training became a standard empirical approach for detecting overfitting. Development of the theorems discussed in this chapter lends theoretical underpinning to experimental evidence supporting intuitively satisfying concepts.

The convergence theorem discussed in this chapter is formally stated and proven in the Appendix along with other supporting theorems building up to it. This chapter explains the development of the convergence theorem conceptually only – the rigorous proof appears in the Appendix.

9.1 Similar convergence theorems

The scenario of solving BVPs discussed in the previous section (and formalized in the next section) is not unique to ANN methods; any number of techniques could be imagined which iteratively adjust parameters by observing how well the DE is satisfied within the domain. By intuitive argument, any such method which satisfies the BCs and reduces error in the DE everywhere in the domain during every iteration should converge to the correct BVP solution. Numerical solution of BVPs is a field of such importance that one would expect convergence in this general case to have been studied previously. However, it would appear that development of the leading technique for numerical solution of BVPs, the finite element method, has directed consideration of convergence issues in a different direction.

The FEM in its basic form is not an iterative technique. The BVP is discretized and converted into a system of equations. The resulting system will be linear whenever the DE in question is also linear. In such a case, iterative techniques are not generally required since the system can be solved directly. Despite solving in a single "iteration", this technique is still concerned with convergence. However, convergence in this sense refers to obtaining the desired exact solution as the grid size approaches zero [48] rather than as the number of iterations approaches infinity.

As popularity of the FEM grew, so did the size of linear systems being considered. Memory requirements and computing speed stymied efforts to solve large systems of equations, and iterative techniques with lower computional overhead were considered. The convergence theorems for several different iterative methods [49-53] rely upon proving that the spectral radius of the matrix operator used is less than unity.

The system of equations produced by the FEM for nonlinear DEs will also be nonlinear, resulting in the necessary application of iterative solution techniques. Early methods for solution of nonlinear BVPs included gradient descent techniques [48] which are well known to converge to the correct solution only when the original approximation is sufficiently accurate. More modern iterative techniques such as the multigrid method [54, 55] often explore some form of spectral radius when proving convergence.

From its inception, the FEM has been concerned with converting BVPs into system of equations. Research in numerical solution of BVPs correspondingly focused upon techniques for solving the resulting systems rather than considering satisfaction of the DE directly. For this reason, no convergence theorem similar to the one presented in this work could be found in the literature.

One might instead consider recasting the ANN method proposed here in such a manner that the many convergence theorems from FEM theory might be applied. The resulting system of equations would contain the ANN weights $\boldsymbol{\theta}$ as the unknowns rather than values of the approximate solution. It is not immediately clear whether Green's Theorem could even be applied to discretize the BVP after applying the required DE to the approximate solution in (99) which contains nested exponentials of all the ANN weights represented by the network output $N(\boldsymbol{\theta})$. If discretization is possible, each

equation in the system would consist of a complicated and highly nonlinear DE involving all of the weights in θ. The matrix resulting from the system of equations in this case would not be sparse, a property exploited in many FEM algorithms. Finally, performance of FEM iterative techniques is highly sensitive to discretization and the underlying DE [55] even without considering the additional complication of applying them to the approximate solution in (99). The combined challenge of these obstacles directed effort towards developing the convergence theorem in this work rather than applying existing FEM convergence theory to the ANN method.

9.2 Problem definition

The theorems referred to in this chapter use essentially the same definition of the DE and of the boundary geometry and its conditions as presented earlier. Several other requirements are necessary as well. For example, each boundary segment is defined by a parameterized function with parameter s representing location on the boundary. At least one segment of non-zero length must have a Dirichlet boundary condition where the value $\psi(s)$ is specified (the "anchor"). Those boundary segments without Dirichlet conditions are instead specified by Neuman conditions $\hat{\mathbf{n}}(s)\cdot\nabla\psi(s)=g(s)$ where $\hat{\mathbf{n}}(s)$ is the inwardly-directed unit vector normal to the boundary at s, and the function $g(s)$ is specified. The introduction of the parameter s simply redefines the boundary and its value parametrically rather than with the Cartesian vector $\mathbf{x}$.

The BVP must be solved numerically using an iterative technique (such as the RPROP or Levenberg-Marquardt algorithms) where the TAS ψ_t is continuous everywhere inside the domain and exactly satisfies all boundary conditions during all iterations, such as in (99). The change in the TAS in subsequent iterations is assumed to

be sufficiently small to be considered differential in magnitude. In principle, the amount ANN parameters change during an iteration can be scaled so that this is true to an arbitrary degree. Quantities and functions undergoing small but finite changes during ANN training are indicated using the δ symbol, i.e. a quantity a changes by an amount δa during training.

9.3 Important notation

The error in the DE solution $D(\mathbf{x})$ at a given point $\mathbf{x}$ in the domain, recalled from (2), is of obvious import for gauging performance of the TAS. In general, the value of D is unknown since the analytical solution is not available – indeed, no need would exist for developing the TAS were the analytical solution already known. Consequently, the TAS will be produced by reducing the error in the DE in (1), i.e. the difference between the left- and right-hand sides of (1), rather than the error defined by (2). The left-hand side of (1) generally contains terms involving the various partial derivatives of the analytical solution, and so errors in these derivatives will also be important. First-derivative error is a vector quantity involving gradients and is defined as

$$\mathbf{D}'(\mathbf{x}) \equiv \nabla \psi_t(\mathbf{x}) - \nabla \psi(\mathbf{x}) \tag{124}$$

In Cartesian coordinates, (124) becomes

$$\mathbf{D}'(x,y) \equiv \frac{\partial \psi_t}{\partial x}\hat{\mathbf{i}} + \frac{\partial \psi_t}{\partial y}\hat{\mathbf{j}} - \frac{\partial \psi}{\partial x}\hat{\mathbf{i}} - \frac{\partial \psi}{\partial y}\hat{\mathbf{j}} = \left(\frac{\partial \psi_t}{\partial x} - \frac{\partial \psi}{\partial x}\right)\hat{\mathbf{i}} + \left(\frac{\partial \psi_t}{\partial y} - \frac{\partial \psi}{\partial y}\right)\hat{\mathbf{j}} \tag{125}$$

for the two-dimensional special case, which may be simplified to

$$\mathbf{D}'(x,y) = D'_x\hat{\mathbf{i}} + D'_y\hat{\mathbf{j}} \tag{126}$$

The quantities $D'_x \equiv \partial\psi_t/\partial x - \partial\psi/\partial x$ and $D'_y \equiv \partial\psi_t/\partial y - \partial\psi/\partial y$ in (126) are the errors in the x and y components of the gradient, respectively. Error in the second derivative is similarly defined as

$$\mathbf{D}''(x,y) = D''_x\hat{\mathbf{i}} + D''_y\hat{\mathbf{j}} \text{ where } D''_x \equiv \frac{\partial^2\psi_t}{\partial x^2} - \frac{\partial^2\psi}{\partial x^2} \text{ and } D''_y \equiv \frac{\partial^2\psi_t}{\partial y^2} - \frac{\partial^2\psi}{\partial y^2} \tag{127}$$

Finally, two useful definitions are

$$D' \equiv D'_x + D'_y \text{ and } D'' \equiv D''_x + D''_y \tag{128}$$

9.4 Explanation of the convergence theorem

The convergence theorem begins somewhat indirectly by assuming that $|D'| = |D'_x + D'_y|$ is reduced during training. This requirement would be a logical assumption when solving the DE

$$\frac{\partial\psi}{\partial x} + \frac{\partial\psi}{\partial y} = 0 \tag{129}$$

but has little obvious bearing on solving other, more complicated DEs. A progression of theorems begins with this assumption and first proves that reducing $|D'_x + D'_y|$ everywhere in the domain guarantees that the magnitude of the solution error D will also be reduced everywhere in the domain. Later theorems then generalize this result to encompass essentially any DE.

The first five theorems in the Appendix establish general properties of real numbers which are used in the following theorems. Theorem 6 then proves that reducing $|D'_x + D'_y|$ implies that at least one of either $|D'_x|$ or $|D'_y|$ (and sometimes both) will also be reduced. Theorem 7 involves appropriately defining a path between any two points in

the domain and applying Theorem 6 to ensure that $|d\mathbf{x}\cdot\mathbf{D}'|$ is reduced everywhere along the path. The quantity $d\mathbf{x}\cdot\mathbf{D}'$ is the amount of error introduced into the TAS when moving a differential distance $d\mathbf{x}$ from a particular point in the domain. To understand why this is so, first consider a point (x_0, y_0) lying on a Dirichlet boundary segment as in Figure 40.

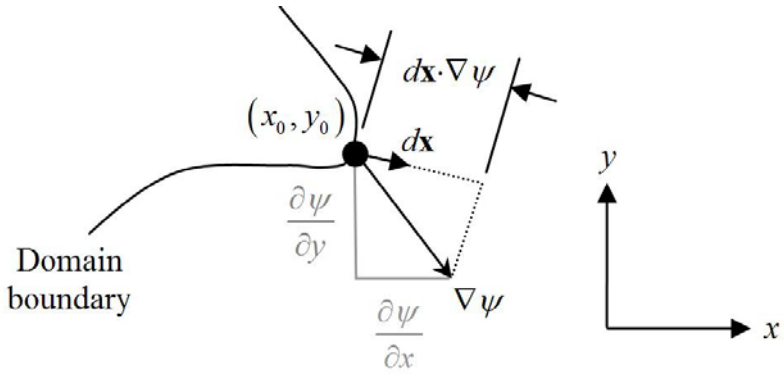


Figure 40. Geometry involved when taking a step $d\mathbf{x}$ from a point (x_0,y_0) on the domain boundary.

The direction of the gradient $\nabla\psi$ in Figure 40 will depend on the local shape of the true solution. Before considering moving from (x_0,y_0) in an arbitrary direction $d\mathbf{x}$, first assume the special case that $d\mathbf{x}$ corresponds with the x-direction. Figure 41 displays a cross-section of the true and approximate solutions starting at the point (x_0, y_0) and continuing in the x-direction. Both the true solution ψ and the TAS ψ_t will be coincident at (x_0, y_0) since the TAS is exact at the domain boundary. The slopes of the cross-section curves represent the partial derivatives with respect to x, and the values of the trial and true solutions a differential distance dx from point (x_0,y_0) are

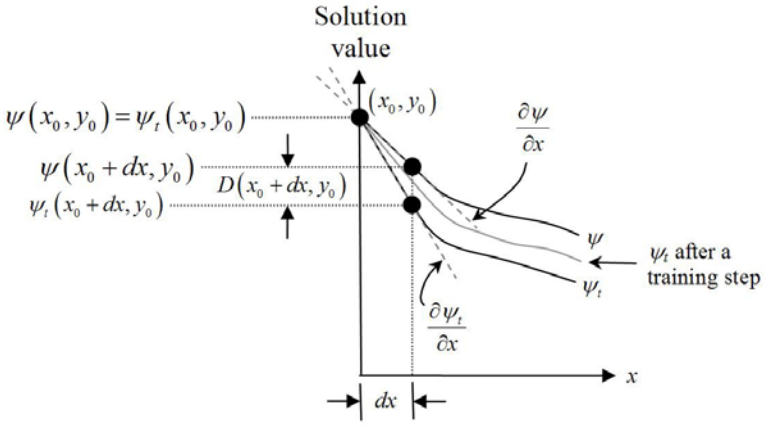


Figure 41. Cross-sections of the true and approximate solutions in the *x*-direction.

$$\psi(x_0 + dx, y_0) = \psi(x_0, y_0) + dx \left.\frac{\partial \psi}{\partial x}\right|_{(x_0, y_0)} \tag{130}$$

and

$$\psi_t(x_0 + dx, y_0) = \psi_t(x_0, y_0) + dx \left.\frac{\partial \psi_t}{\partial x}\right|_{(x_0, y_0)} \tag{131}$$

respectively. Subtracting (131) from (130) produces

$$D(x_0 + dx, y_0) = D(x_0, y_0) + dx D'_x(x_0, y_0) \tag{132}$$

when applying the notation defined earlier. The value of $D(x_0, y_0)$ is of course zero, but remains in (132) for consideration of a more general case where (x_0, y_0) is not on the domain boundary.

Now consider the TAS after the ANN undergoes a training step, whose approximate solution is represented by the lighter curve in Figure 41. This curve is drawn assuming that $\partial \psi_t / \partial x$ has improved during training, that is, is closer in magnitude to the true $\partial \psi / \partial x$. This is equivalent to stating that $|D'_x|$ has become smaller during training. Recall that Theorem 6 guarantees that either $|D'_x|$ or $|D'_y|$ will be

reduced everywhere in the domain during training. Figure 41 can be redrawn using the *y*-axis instead if $\left|D'_y\right|$ and not $\left|D'_x\right|$ was reduced. It is now clear from (132) that the local solution error $D\left(x_0+dx, y_0\right)$ has been reduced during training. The quantity dxD'_x is now apparent as the error introduced into the TAS when moving from (x_0,y_0) in the *dx* direction.

Now consider taking another step in the *dx* direction. The local value of *D* at the start of this step is not zero since the point no longer lies on the domain boundary. Even so, the local value of dxD'_x will again indicate an additional error contribution associated with the current step, indicating that (132) is valid for arbitrary points (x_0, y_0). Reducing the quantity $\left|dxD'_x\right|$ is of obvious import for obtaining more accurate solutions through training.

The goal is now to define a path between any two arbitrary points and show that $\left|dxD'_x\right|$ (or alternatively $\left|dxD'_y\right|$ if moving in the *y*-direction) is reduced everywhere along that path. To clarify this task, consider that a training iteration has just been completed. The TAS has been adjusted by a differential amount everywhere in the domain, and training is paused to search for a viable path. Theorem 6 guarantees that either $\left|D'_x\right|$ or $\left|D'_y\right|$ (and sometimes both if lucky) will have been reduced at every location in the domain during the just completed training iteration.

Starting at one of the two arbitrary points, a step can be taken towards the other point in either the *x*- or *y*-direction, depending on whether $\left|D'_x\right|$ or $\left|D'_y\right|$ was reduced during the previous training iteration (positive or negative directions are chosen as

appropriate depending on the relative locations of the two arbitrary points). In the fortuitous event when both quantities were reduced, a choice can be made as to whether moving in the *x*- or *y*-direction would be more beneficial for approaching the second point. However, the behavior of $\left|D_x'\right|$ and $\left|D_y'\right|$ is completely dependent on the details of the just previous training iteration, and no obvious guarantee exists that any two points can be connected by a viable path when only traveling in the cardinal directions.

Return now to Figure 40 and consider allowing arbitrary directions for $d\mathbf{x}$. A figure analogous to Figure 41 could be created where the cross-section now appears in the arbitrary $d\mathbf{x}$ direction rather than the dx direction. The counterpart to (132) when using a $d\mathbf{x}$ step is then

$$D\left(\mathbf{x}_0 + d\mathbf{x}\right) = D\left(\mathbf{x}_0\right) + d\mathbf{x} \cdot \mathbf{D}'\left(\mathbf{x}_0\right) \tag{133}$$

The quantity of interest is now $d\mathbf{x} \cdot \mathbf{D}'$, which quantifies the error introduced into the TAS when taking the differential step $d\mathbf{x}$.

Consider investigating the error $D\left(\mathbf{x}_n\right)$ at an arbitrary point $\mathbf{x}_n$. Theorem 7 proves that a path exists from any point $\mathbf{x}_0$ to $\mathbf{x}_n$ where $\left|d\mathbf{x} \cdot \mathbf{D}'\right|$ is reduced during training for each of the n differentially close points composing the path. Such a path appears in Figure 42 with the point $\mathbf{x}_0$ chosen to lie on the domain boundary so that $D\left(\mathbf{x}_0\right) = 0$. The solution error at point $\mathbf{x}_2$

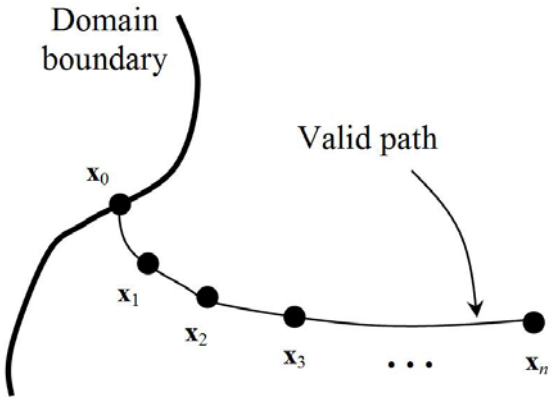


Figure 42. A valid path consisting of *n* differentially close points from the domain boundary to arbitrary point $\mathbf{x}_n$ where $d\mathbf{x}\cdot\mathbf{D}$ is reduced during training everywhere along the path.

$$D(\mathbf{x}_2) = D(\mathbf{x}_1) + d\mathbf{x}\cdot\mathbf{D}'(\mathbf{x}_1) = D(\mathbf{x}_0) + d\mathbf{x}\cdot\mathbf{D}'(\mathbf{x}_0) + d\mathbf{x}\cdot\mathbf{D}'(\mathbf{x}_1) \tag{134}$$

is calculated by applying (133) recursively. The recursive definition for solution error can be extended to point $\mathbf{x}_n$ to produce

$$D(\mathbf{x}_n) = D(\mathbf{x}_0) + \sum_{i=0}^{n-1} d\mathbf{x}\cdot\mathbf{D}'(\mathbf{x}_i) = \sum_{i=0}^{n-1} d\mathbf{x}\cdot\mathbf{D}'(\mathbf{x}_i) \tag{135}$$

Solution error is expressed in (135) as the sum of $d\mathbf{x}\cdot\mathbf{D}'$ along a path for which $\left|d\mathbf{x}\cdot\mathbf{D}'\right|$ is reduced during training everywhere along the path. Theorem 8 states that the magnitude solution error $D(\mathbf{x}_n)$ must be smaller in subsequent training iterations if this is the case. These theorems prove that the solution error must converge toward zero as long as $\left|D'_x + D'_y\right|$ continues to be reduced everywhere in the domain during training.

The requirement on $\left|D'_x + D'_y\right|$ is a reasonable assumption when solving the DE in (129). Theorem 9 extends this result by drawing an analogy between the assumption on $\left|D'_x + D'_y\right|$ and the resulting implication for $|D|$; it is assumed instead that $\left|D''_x + D''_y\right|$ is reduced which leads to the implication that $\left|D'_x + D'_y\right|$ must also be reduced. Theorem 10

then applies Theorem 9 to show that the TAS will converge to the true solution when solving the nonhomogenous Laplace equation, provided that error in the DE is reduced everywhere in the domain during training. The Laplace equation is, of course, the appropriate analogue to (129) for Theorem 10.

The final convergence theorem appears in Theorem 11 which considers more general DEs. Convergence is guaranteed whenever the error in the DE is reduced everywhere during training for any nonhomogenous DE composed of monotonic functions of any partial derivatives of ψ. The monotonic requirement ensures that either $\left|D'_x\right|$ or $\left|D'_y\right|$ will be reduced even for complicated DEs. This monotonic requirement appears restrictive, but in fact the functions must be monotonic only in certain ranges. The required ranges are dependent on the degree of similarity between the true and approximate solutions. An approximate solution with low error will demand a smaller range, and thus the monotonic requirement will be fulfilled for TASs with reasonable accuracy regardless of the DE. The result is a convergence theorem which can, for all practical purposes, be applied to any DE.

9.5 Empirical evidence for the convergence theorem

A reduction in the error in the solution is assured by Theorem 11 when error in the DE reduces at every location within the domain. Certainly, DE error will not always become smaller at every location in the domain during every training iteration. However, a strong correlation has been observed between the RMS error in the DE, E from (5), and the error norm from (57). Both E and E_{norm} are evaluated over a set of points T. The ANN weights are updated during training using E evaluated with a given training set. Generalization performance for the ANN solution is better gauged using a

different set, including different and more finely sampled points – usually on a grid at least three times finer than the training grid.

Figure 43 shows the evolution during training of $\sqrt{E}$ and E_{norm} for solution of the nonlinear, nonhomogenous DE in (118) treated in [29] which has the exact analytical solution (119) over the star-shaped domain with mixed Dirichlet and Neuman boundary conditions. The continuous curve in Figure 43 is evaluated only at training points, whereas the points represented by discrete symbols are evaluated with the finer grid. The DE error E for only the training points does not produce a smooth curve; the discontinuous jumps occur at times during training when points are added to the training set [29]. Additionally, E evaluated with only training points continues to decrease even after the error norm has leveled out and begins to rise slightly at the onset of overfitting. The fitness of the DE error over the entire domain is better represented by E evaluated on the finer grid, and not only at training points; the progression of square markers in Figure 43 closely follows that of the error norm with triangular markers, accurately predicting when the benefits of continued training have ended and the onset of overfitting. Remember that error norm values are not available during training since the true solution is unknown. This leaves E as the only metric to gauge effectiveness of the ANN solution and indicate when training should be halted.

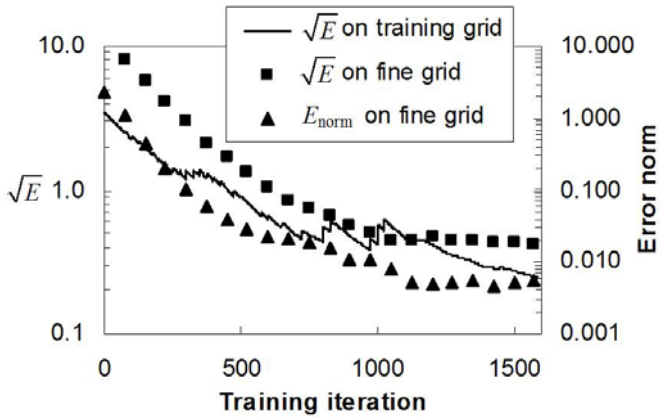


Figure 43. Progression of $\sqrt{E}$ and E_{norm} in the solution during training of the ANN solution.

9.6 Conclusion of the convergence theorem

Theorem 11 is important for two reasons. First and most importantly, it assures the user that convergence toward satisfaction of the DE directly implies convergence toward the correct solution. This relationship is essential for acceptance of ANNs as a reliable method for the numerical solution of DEs. The second reason lies in theoretical support for an observed phenomenon. Good TASs have been observed to be generated when the error in the DE over the domain is smooth and relatively flat, especially between training points, and decreases in magnitude through successive iterations. In fact, observing changes in DE error over the domain from iteration to iteration has been instrumental in determining the optimal number of hidden nodes and placement of training points, yielding a reliable metric for detecting the onset of overfitting. The theorem confirms this practice by establishing theoretically what has been observed during numerical experiments.

10 Software implementation

The results reported in this work were generated using code developed completely by the author in an object-oriented C++ environment. The three classes `Network`, `DiffEQ`, and `BoundaryCondition` appearing in the class diagram in Figure 44 are the coded representations of ANNs, DEs and BCs, respectively. Objects from the `DiffEQ` class contain one instance of the `BoundaryCondition` class for each boundary segment.

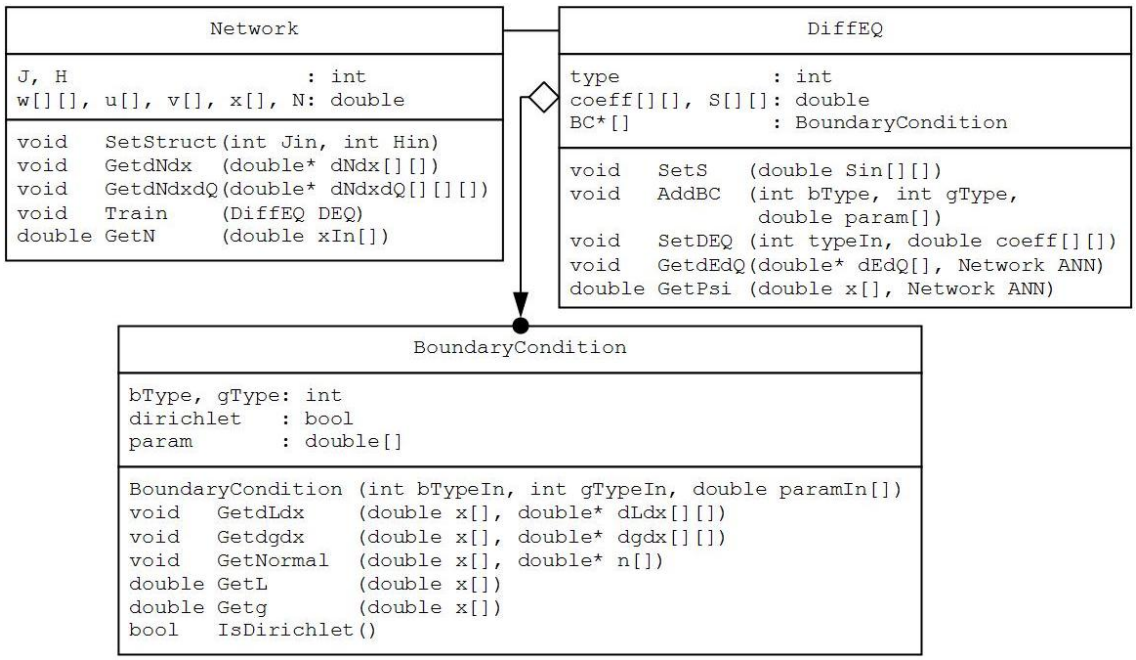


Figure 44. Class diagram for implementation of ANNs for solving BVPs.

Objects for boundary segments are either `dirichlet` or not, i.e. Neuman, which is returned by `IsDirichlet`. The function type, such as quadratic, circular, etc., defining the shape of a boundary segment is indicated by `bType`, just as the function

type for the BC value function g_i is stored in `gType`. The details of these function types, such as radius for a circle for example, are stored in the `param` array and are initialized in the object constructor. The methods `GetL` and `Getg` return L_i and g_i given the vector **x** which is passed as the argument `x[]`. The partial derivatives $\partial^k L_i / \partial x_j^k$ evaluated at a given **x** are returned by `GetdLdx` in the variable `dLdx[][]` where one of the array dimensions represents j and the other k. A similar method returns the derivatives $\partial^k g_i / \partial x_j^k$. The components n_{ij} of the normal to the boundary segment at **x** is returned in the variable `n[]` by the method `GetNormal`.

The `Network` and `DiffEQ` classes are related to one another since methods in one class require access to methods from the other class. The access issue could be solved by including an instance of one class as a member of the other. For example, an object representing the DE to be solved could be a member of the ANN object which will be optimized to solve it. However, the DE object member would need to be replaced when examining the effects of weight reuse; the ANN object would first be trained to solve a particular DE and then the DE object would need to be modified to the second DE before evaluating the effects of weight reuse from solving the first DE. Instead, an ANN object can be created along with any number of DE objects in the `main` program. Then, any given DE object can be passed as an argument to ANN methods requiring it, and vice versa.

Solution of a DE with this approach begins by creating an instance of the `Network` class and indicating with `SetStruct` how many hidden nodes `H` it has and its number of inputs `J`, which corresponds to the size of the vector **x**. The DE to be solved is stored in a `DiffEQ` object where each boundary segment is added with the method

`AddBC` by specifying the details of each desired segment. The form of the DE is specified in `SetDEQ` by the coefficients on the various $\partial^k \psi / \partial x_j^k$ terms and the `type` of the function composing the non-homogenous term. Finally, the points for the set `S` where the error function in (22) is to be evaluated is passed to the DE object through `SetS`.

The initialized `DiffEQ` object is then passed as an argument to the `Train` method in the `Network` object to optimize the weights for solving that particular DE. The `Train` method will then obtain the error gradient $\partial E / \partial \mathbf{\theta}$ by calling `GetdEdQ` from the `DiffEQ` object. `GetdEdQ` in turn requires the ANN output from `GetN` and the partial derivatives $\partial^k N / \partial x_j^k$ and $\partial^{k+1} N / \partial x_j^k \partial \theta_i$ from `GetdNdx` and `GetdNdxdQ`, respectively. The point **x** is stored in the ANN object when an output is calculated by `GetN` so that subsequent calls to `GetdNdx` and `GetdNdxdQ` know at which point to evaluate the partial derivatives. The hidden node weights `w[][]`, hidden node biases `u[]`, and output node weights `v[]` are also stored in the ANN object so that a call to `GetPsi` in a `DiffEQ` object can return the TAS ψ_t at any `x[]` once the ANN is trained. Any instances of `Network` and `DiffEQ` objects can be used interchangeably in the `main` program as long as they have the same number of inputs, as stored in `J` and `coeff[][]`.

11 Conclusions and recommendations

The trial approximate solution in (99) is the culmination of a concentrated effort to define a boundary value problem solution which not only satisfies all Dirichlet and/or Neuman boundary conditions exactly, but also is free of singularities which hinder the ANN from obtaining an accurate TAS within the domain. When this project was begun, several topics were enumerated which would be documented in the final dissertation:

- *Investigate alternative learning algorithms for updating ANN weights*. The learning algorithm RPROP was originally chosen to update weights, but was later replaced by the more efficient Levenberg-Marquardt algorithm. Both algorithms are defined in Chapter 2.
- *Explore the benefits of weight reuse when solving different BVPs*. This is the only topic not receiving as much attention as originally intended. Chapter 5 details this effort and describes how other interesting topics shifted focus away from this one.
- *Test BVPs with mixed Dirichlet/Neuman conditions on irregular boundaries*. This topic proved more difficult than originally estimated. Chapter 6 documents the evolution of the final TAS in (99) from an earlier TAS for mixed conditions which was mathematically valid but difficult to obtain accurate results with in practice.
- *Further examine the effect of length factor definition on validity of the TAS*. Chapter 7 is devoted to length factor definition, presenting a comparison of different length factors and their impact on solution fitness. Judicious definition of length factors represents a significant area where the user can positively affect performance by incorporating a priori knowledge of the problem.

- *Confirm successful solution of more complicated BVPs*. Chapter 8 presents numerical solution of several BVPs on irregular domains. These BVPs include nonlinear and non-homogenous differential equations. All problems considered to date have been solved to an error norm on the order of 10^{-2} or lower, sufficient for most engineering applications.
- *Document the relationship between error in the DE and error in the TAS*. The intuitive correlation between fitness in the approximate solution and satisfaction of the DE is confirmed empirically through numerical experiments. These observations are in turn confirmed theoretically by a theorem in Chapter 9 stating that residual error in the approximate solution will decrease everywhere in the domain given that satisfaction of the DE similarly improves during training of the ANN.

The practical milestones set at the beginning of this project have been met, while at the same time pursuing the author's overarching research goal of developing intelligent systems which make use of previous knowledge to aid in solving new problems. Knowledge of the BVP solution is encoded into the system via automatic satisfaction of BCs and domain geometry through the length factors. The ANN then starts not at a random location in the search space, but at one making use of a priori knowledge of the problem. The result is an ANN method for solving BVPs significantly less complicated, and often more accurate, than successful ANN methods in the literature.

The simplicity of the method has the added benefit of making it more accessible to the inexperienced ANN user. The standard multilayer perceptron structure used is the most prevalent of any ANN architecture, and the Levenberg-Marquardt learning

algorithm is a well-documented classic gradient-descent optimization approach. While ANN methods for solving BVPs are not currently sufficiently mature to rival commercial FEM software packages, some users will find ANN approaches desirable. Benefits such as exact BC satisfaction, consideration of complicated non-standard DEs, a continuous and differentiable approximate solution, and elimination of meshing concerns may entice some users to try this method, and its inherent simplicity will not deter them.

Figure 1 illustrates the process involved in solving a BVP with the proposed method. Some steps in the process involve actions to be taken by the user. As with any ANN problem, the number of hidden nodes and the location of training points must be determined. The user must break the domain boundary into an arbitrary number of segments and define a length factor for each segment. The standard length factor definition can be used when no knowledge of solution behavior exists. Incorporating any a priori knowledge of the solution behavior can, however, greatly improve performance as documented in Chapter 7. Continuous functions for each segment's BC and normal vector must also be defined. These functions are generally straightforward to define on the basis of boundary shape and boundary conditions given in the problem statement.

The actions required by the user for solving BVPs are sufficiently simple that nominal experience using the method enables the user to efficiently prepare new BVPs for accurate solution by the ANN. The work required of the user could be reduced further by automating choices of design parameters. An algorithm to accomplish this could, for example, dynamically reduce grid size if satisfaction of the DE is observed to

worsen between training points, or dynamically increase the number of hidden nodes if error in the DE is not sufficiently reduced. Such an algorithm would make this method more attractive to users who wish to incorporate a priori knowledge of solution behavior into length factor definitions, but have little or no experience using ANNs.

Example BVPs solved in this dissertation are limited to two dimensions out of simplicity of implementation rather than any conceptual difficulty. In fact, all equations are derived with a spatial vector **x** of arbitrary dimension (the Appendix presents theorems with a two-dimensional **x** in order to facilitate understanding of the topic). The greatest obstacle in implementing a three-dimensional BVP lies in determination of the various partial derivatives required for the error gradient as presented in Chapter 6.7. These partial derivatives are straightforward if time-consuming to develop, but can be completely reused for any subsequent third-dimensional problem. The availability of symbolic derivative evaluation such as Mathmatica® render this step less daunting, however.

Even with determination of the required error gradient partial derivatives, the trial approximate solution in its current form (99) is defined only for Dirichlet and Neuman BCs. The TAS must be modified to consider second-order BCs. Extension of the TAS from Dirichlet to Neuman conditions proved more difficult than originally believed. Consideration of second-order BCs has not yet been investigated, but surely lessons learned in extending to Neuman conditions would ease appropriate definition of a TAS for all three types of BCs.

Another topic for future research involves a return to the original concept of how weight reuse can improve efficiency of BVP solution development. This dissertation has

concentrated primarily on innovative development of an ANN method exactly satisfying BCs on irregular domains, something not previously appearing in the literature. With this obstacle overcome, one can imagine a system employing weight reuse to quickly and accurately solve new BVPs by reapplying weights from solutions to previously considered BVPs similar to the new BVP. Such a system would make use of previous knowledge of the solution *without* intervention by the user. Infusion of knowledge via BCs and length factors, although a significant improvement over the usually stochastic choice of search space starting location, is something dependent on control by the user. Application of weight reuse to build a system more powerful at solving BVPs for every new BVP solved represents a giant step towards developing a system truly capable of learning independent of human interaction.

While the ANN method for numerical solution of BVPs documented in this dissertation does not immediately rival the established FEM, it does provide an alternative with certain unique properties desirable to certain users. For example, a priori knowledge of solution behavior can be incorporated into length factor definitions, grid meshing is not required, interpolation between training points is automatic and extremely accurate, and complicated nonlinear DEs impose no additional modeling complexity. At the same time, the proposed method shows great promise, with continued research, to grow into a field which eventually could press the frontier of artificial learning while providing a powerful new tool available to engineers and scientists for solving BVPs.

Appendix

Chapter 9 discusses several theorems which are presented rigorously here. Theorems 1 through 5 establish general relationships between real numbers, while Theorems 6 through 11 are specific to the solution of BVPs with ANNs. The later theorems refer to the problem definition as defined in Chapter 9.

Theorem 1

Statement

Consider that the scalar quantities a and b undergo arbitrarily small changes from values a_1 and b_1 to $a_2 = a_1 + \delta a$ and $b_2 = b_1 + \delta b$. If $|a_2| < |a_1|$ and $|b_2| < |b_1|$, then the only possibility for which $|a_2 + b_2| > |a_1 + b_1|$ is if sgn $a_1 \neq$ sgn b_1 and one of the following is true

- either $|\delta a| < |\delta b|$ in the case $|a_1| > |b_1|$
- or $|\delta b| < |\delta a|$ in the case $|b_1| > |a_1|$

Proof

The results of this theorem are summarized in Table 3. If the magnitude of a_1 becomes smaller, then the sign of a and the sign of its change δa must be opposite; the same can be said for the signs of b_1 and δb. If the quantities a and b have the same sign and both undergo small changes which preclude them from changing sign, then $|a_2 + b_2| < |a_1 + b_1|$ regardless of the values for a_1, b_1, δa and δb.

Table 3. Summary of Theorem 1.

Requirements	Main case	Sub-case	Result
$\operatorname{sgn} a_1 \neq \operatorname{sgn} \delta a$, $\operatorname{sgn} b_1 \neq \operatorname{sgn} \delta b$	$\operatorname{sgn} a_1 = \operatorname{sgn} b_1$	All sub-cases	$\lvert a_2 + b_2 \rvert < \lvert a_1 + b_1 \rvert$
	$\operatorname{sgn} a_1 \neq \operatorname{sgn} b_1$	$\lvert a_1 \rvert > \lvert b_1 \rvert, \lvert \delta a \rvert > \lvert \delta b \rvert$	$\lvert a_2 + b_2 \rvert < \lvert a_1 + b_1 \rvert$
		$\lvert b_1 \rvert > \lvert a_1 \rvert, \lvert \delta b \rvert > \lvert \delta a \rvert$	$\lvert a_2 + b_2 \rvert < \lvert a_1 + b_1 \rvert$
		$\lvert a_1 \rvert > \lvert b_1 \rvert, \lvert \delta a \rvert < \lvert \delta b \rvert$	$\lvert a_2 + b_2 \rvert > \lvert a_1 + b_1 \rvert$
		$\lvert b_1 \rvert > \lvert a_1 \rvert, \lvert \delta b \rvert < \lvert \delta a \rvert$	$\lvert a_2 + b_2 \rvert > \lvert a_1 + b_1 \rvert$

Four sub-cases exist when the signs of a_1 and b_1 are different. The sign of $a_1 + b_1$ will be determined by the sign of the quantity with the largest magnitude, which is the same case for the sign of $\delta a + \delta b$. The quantities $a_1 + b_1$ and $\delta a + \delta b$ must have different signs when $|a_2 + b_2| = |a_1 + b_1 + \delta a + \delta b| < |a_1 + b_1|$, which requires that the larger of the two magnitudes between a_1 and b_1 must also undergo the larger of the two changes. Therefore the only possibilities for $|a_2 + b_2| > |a_1 + b_1|$ appear in the theorem statement.

Theorem 2

Statement

If a is a constant and the quantity b undergoes an arbitrarily small change from b_1 to b_2 such that $|a + b_2| < |a + b_1|$, then one of the following must be true:

- either $\lvert b_2 \rvert < \lvert b_1 \rvert$
- or $\lvert b_2 \rvert > \lvert b_1 \rvert$, $\lvert a \rvert > \lvert b_1 \rvert$ and $\operatorname{sgn} a \neq \operatorname{sgn} b_1$

Proof

If $\operatorname{sgn} a = \operatorname{sgn} b_1$, then the only way to reduce $|a + b_1|$ is if $|b_2| < |b_1|$. If $\operatorname{sgn} a \neq \operatorname{sgn} b_1$ and $|a| < |b_1|$, then $\operatorname{sgn}(a + b_1) = \operatorname{sgn} b_1$. In this case, a small change from b_1 to b_2 with $|b_2| > |b_1|$ would cause $|a + b_2| > |a + b_1|$. This contradicts the assumption that $|a + b_2| < |a + b_1|$, leading to only the two possibilities in the theorem statement.

Theorem 3

Statement

Consider that the quantities a and b undergo an arbitrarily small change from values a_1 and b_1 to a_2 and b_2. If $|a_2 + b_2| < |a_1 + b_1|$, then one of the following must be true:

- either both $|a_2| \leq |a_1|$ and $|b_2| < |b_1|$
- or either $|a_2| < |a_1|$ or $|b_2| < |b_1|$
- or both $|a_2| \geq |a_1|$ and $|b_2| > |b_1|$, and $\operatorname{sgn} a_1 \neq \operatorname{sgn} b_1$

Proof

Reducing $|a+b|$ requires moving closer to the line $|a+b|=0$. Figure 45 illustrates six points in various regions of *a*-*b* space where the dotted lines indicate lines of constant $|a+b|$ through each point. Moving in any direction indicated by the semicircular sectors will thus produce a smaller value of $|a+b|$. Consider a point in the first quadrant where both a and b are positive. The semicircular sector of permissible directions is divided into three regions by the dashed lines: a is reduced above the horizontal, b is reduced to the right of the vertical, and both a and b are reduced for the region in between. In all cases, one or both of the quantities a and b must be reduced to reduce $|a+b|$. A similar argument is made for points in the third quadrant by considering $|a|$ and $|b|$ rather than simply a and b; again one or both of the quantities $|a|$ and $|b|$ must be reduced to reduce $|a+b|$. Permissible directions for points in the second and fourth quadrant are also divided into three regions: one where either $|a|$ or $|b|$ is reduced, one where both $|a|$ and $|b|$ are reduced, and the shaded regions where both $|a|$ and $|b|$ increase. Therefore, one of the three cases in the theorem statement must be true for every point in

a-*b* space. Notice that the first and third cases in the theorem statement involve one inequality which is strict and one which is not. When undergoing small changes, one of the quantities of *a* and *b* may possibly remain constant, but the other must change since the magnitude of the sum of them must decrease. Whether the strict inequality appears in the *a* or *b* equation is arbitrary, and is not important since *a* and *b* can be interchanged due to their symmetrical relationship.

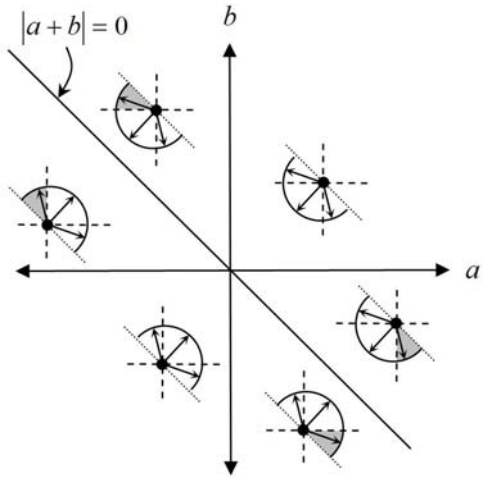


Figure 45. Possible directions for reducing |*a*+*b*| at various locations in *a*-*b* space. Shaded regions indicate directions for which both |*a*| and |*b*| increase.

Theorem 4

Statement

Consider that both *a* and *b* undergo arbitrarily small changes from values a_1 and b_1 to $a_2 = a_1 + \delta a$ and $b_2 = b_1 + \delta b$. If $|a_2 + b_2| < |a_1 + b_1|$ and $a + mb = 0$, then $|a_2| < |a_1|$ and $|b_2| < |b_1|$ for any value of *m* other than unity.

Proof

The theorem statement implies that

$$a_2 + mb_2 = a_1 + \delta a + mb_1 + m\delta b = \left(a_1 + mb_1\right) + \left(da + m\delta b\right) = \delta a + m\delta b = 0 \quad (136)$$

which leads to

$$\left|a_2+b_2\right|=\left|a_1+\delta a+b_1+\delta b\right|=\left|-mb_1-m\delta b+b_1+\delta b\right|=\left|\left(1-m\right)\left(b_1+\delta b\right)\right| \qquad (137)$$

The theorem statement also requires that

$$\left|a_1+b_1\right|=\left|-mb_1+b_1\right|=\left|\left(1-m\right)b_1\right| \qquad (138)$$

For any value of m other than unity, (137) will be smaller than (138) only if b_1 and db have different signs. Since b undergoes a small change, then it must be true that $|b_2|<|b_1|$. Equations (136) through (138) can be rewritten to eliminate b rather than a and the result would be that $|a_2|<|a_1|$.

Theorem 5

Statement

Consider an x-y space where the quantities a and b are both continuous and single-valued functions of x and y. The functions defining a and b undergo arbitrarily small changes from $a_1(x,y)$ and $b_1(x,y)$ to $a_2(x,y)$ and $b_2(x,y)$ for which $|a_2(x,y)+b_2(x,y)|<|a_1(x,y)+b_1(x,y)|$ for all values of x and y. If there exists a point (x_0,y_0) for which $|a_2(x_0,y_0)|<|a_1(x_0,y_0)|$ and $|b_2(x_0,y_0)|<|b_1(x_0,y_0)|$, then one of the following must be true for all x and y:

- either both $|a_2(x,y)|<|a_1(x,y)|$ and $|b_2(x,y)|\le|b_1(x,y)|$
- or either $|a_2(x,y)|<|a_1(x,y)|$ or $|b_2(x,y)|\le|b_1(x,y)|$

Proof

All that remains is to eliminate the possibility of the third case from Theorem 3 in order to prove this theorem. This will be done by contradiction. Assume first that there exists some (x_1,y_1) for which $|a_2(x_1,y_1)|>|a_1(x_1,y_1)|$, $|b_2(x_1,y_1)|\ge|b_1(x_1,y_1)|$ and

$\operatorname{sgn} a_1 \neq \operatorname{sgn} b_1$. Then draw an arbitrary path between points (x_0, y_0) and (x_1, y_1) in such a manner that the path does not cross itself, as shown in Figure 46. The functions a_1 and a_2 are defined continuously along the path and since $|a_2(x_0,y_0)|<|a_1(x_0,y_0)|$ and $|a_2(x_1,y_1)|>|a_1(x_1,y_1)|$, then there must exist at least one point (x_2,y_2) along the path for which $|a_2(x_2,y_2)|=|a_1(x_2,y_2)|$. In the case of multiple points satisfying this criterion, choose (x_2,y_2) so that it is the closest to (x_1, y_1) along the path. The small change in a_1 which produces a_2 requires that the signs of $a_1(x,y)$ and $a_2(x,y)$ are the same unless either a_1 or a_2 are zero. This fact implies then that $a_1(x_2,y_2)$ and $a_2(x_2,y_2)$, and not simply their absolute values, must be equal. The value of a at (x_2,y_2) is constant and so Theorem 2 can be invoked. The first case of Theorem 3 indicates that $|b_2(x_2,y_2)|<|b_1(x_2,y_2)|$. Since $|b_2(x_1,y_1)|<|b_1(x_1,y_1)|$, a point (x_3,y_3) on the path between (x_2,y_2) and (x_1,y_1) must exist where $b_1(x_3,y_3)=b_2(x_3,y_3)$ by a similar argument as above. The first case in Theorem 2 cannot be true at this point since $|a_2(x_3,y_3)|<|a_1(x_3,y_3)|$ and $|a_2(x_1,y_1)|>|a_1(x_1,y_1)|$ require existence of a point between them where $a_2=a_1$; recall that (x_2,y_2) was chosen to be the closest point to (x_1,y_1) fulfilling this criterion. The only remaining cases are then points (x_2,y_2) and (x_3,y_3) for which $|a_2|=|a_1|$, $|b_2|>|b_1|$ and $\operatorname{sgn} a_1 \neq \operatorname{sgn} b_1$. This results in a new point which is closer to (x_0,y_0) than (x_1,y_1) along the path and fulfills the contradiction criteria. The above argument can be repeated to find another point closer still to (x_0,y_0). This process is repeated ad infinitum until (x_1,y_1) the converges onto (x_0,y_0). No continuous single-valued function for a can satisfy $|a_2|<|a_1|$ and $|a_2|>|a_1|$ at the same point. Therefore the third case in Theorem 3 cannot be true for the assumptions of this theorem.

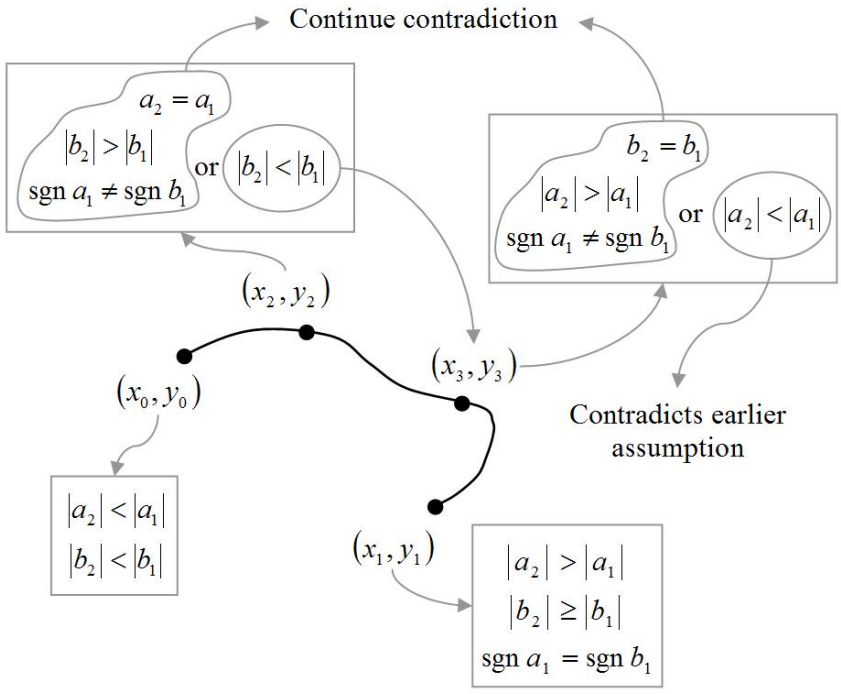


Figure 46. Illustration of the steps involved in proving Theorem 5.

Theorem 6

Statement

At least one of the quantities $|D'_x(x,y)|$ or $|D'_y(x,y)|$ is reduced for all x and y given an ANN solving a differential equation as presented in the problem definition for which $|D'_x(x,y)+D'_y(x,y)|$ is reduced at every point in the domain during a training iteration.

Proof

Theorem 5 guarantees this theorem as long as it can be shown that there exists some point (x_0,y_0) where both $|D'_x(x_0,y_0)|$ and $|D'_y(x_0,y_0)|$ are reduced during the training iteration. Choose any point (x_0,y_0) on a Dirichlet boundary segment with a corresponding distance along the segment s_0. The value of the analytical solution a differential distance away from this point is

$$\psi\left(s_0+ds\right)=\psi\left(s_0\right)+\left[ds\frac{\partial x}{\partial s}\frac{\partial \psi}{\partial x}+ds\frac{\partial y}{\partial s}\frac{\partial \psi}{\partial y}\right]_{s_0} \tag{139}$$

which can also be written in terms of the TAS as

$$\psi\left(s_0+ds\right)=\psi\left(s_0\right)+\left[ds\frac{\partial x}{\partial s}\frac{\partial \psi_t}{\partial x}+ds\frac{\partial y}{\partial s}\frac{\partial \psi_t}{\partial y}\right]_{s_0} \tag{140}$$

since the TAS also satisfies all boundary conditions. Equation (139) is subtracted from (140) to produce

$$\left[ds\frac{\partial x}{\partial s}\left(\frac{\partial \psi_t}{\partial x}-\frac{\partial \psi}{\partial x}\right)+ds\frac{\partial y}{\partial s}\left(\frac{\partial \psi_t}{\partial y}-\frac{\partial \psi}{\partial y}\right)\right]_{s_0}=0\Rightarrow D'_x\left(s_0\right)+mD'_y\left(s_0\right)=0 \tag{141}$$

where m is the slope of the boundary at s_0. Theorem 4 requires that both $|D'_x(x_0,y_0)|$ and $|D'_y(x_0,y_0)|$ must be reduced during the training iteration. The point (x_0,y_0) then fulfills the final requirement for invoking Theorem 5 and so this theorem is proven.

Note that Theorem 4 requires that the boundary slope m is not unity. It is impossible to reduce $|D'_x+D'_y|$ for a unity boundary slope since the two terms always cancel each other to produce exactly zero as required by (141). This represents an exception in the requirement that $|D'_x+D'_y|$ be reduced at every point in the domain; $|D'_x+D'_y|$ will remain zero at such points regardless of changes in the TAS. The quantities D'_x and D'_y represent the errors in $\partial\psi_t/\partial x$ and $\partial\psi_t/\partial y$, respectively, and the magnitudes of these errors (while equal in value and opposite in sign) will converge towards zero during training as $|D'_x+D'_y|$ is reduced at all surrounding points. So even though requiring that $|D'_x+D'_y|$ decreases at such points is not possible (neither can it increase), a requirement where both $\partial\psi_t/\partial x$ and $\partial\psi_t/\partial y$ become closer to their true values is possible. For the purposes of this theorem, the existence of a single point on any

Dirichlet boundary segment where $m \neq 1$ is sufficient. The only possibility for this theorem to fail is if all Dirichlet boundary segments are lines at +45° from the horizontal. If this were true, simply rotating the domain at any angle other than a multiple of 360° would ensure the validity of this theorem.

Theorem 7

Statement

Consider an ANN solving a DE as presented in the problem definition for which $\left|D'_x(x,y) + D'_y(x,y)\right|$ is reduced everywhere in the domain during a given training iteration. A path exists between any two points in the domain for which $|d\mathbf{x}\cdot\mathbf{D}'|$ is reduced during the iteration for all $d\mathbf{x}$ which are tangents to the path.

Proof

This proof involves appropriately defining a path between the arbitrary points (x_0, y_0) and (x, y) by applying Theorem 6 at each point along the path. Theorem 6 requires that either $|D'_x|$ or $|D'_y|$ is reduced during training at all points, a fact which is used to show that $|d\mathbf{x}\cdot\mathbf{D}'|$ is reduced everywhere along the path.

The magnitude of $d\mathbf{x}\cdot\mathbf{D}'$ is the amount of error introduced into the solution by moving the distance $d\mathbf{x}$, and ensuring that it reduces everywhere along the path for a training iteration is requisite to proving that $|D(x,y)|$ is also reduced (which is proven in Theorem 8). Given that $\left|D'_x(x,y) + D'_y(x,y)\right|$ is reduced everywhere in the domain, Theorem 6 requires that either $|D'_x(x,y)|$ or $|D'_y(x,y)|$ (and sometimes both) is similarly reduced everywhere. This fact is then used to define a path from (x_0,y_0) to (x,y) for which $|d\mathbf{x}\cdot\mathbf{D}'|$ reduces everywhere along the path during a training iteration.

Finding such a path requires careful selection of the direction implied by $d\mathbf{x}$ so that $|d\mathbf{x}\cdot\mathbf{D}'|$ reduces. Recall that the reduction of $|d\mathbf{x}\cdot\mathbf{D}'|$ is due to changes to the TAS during training, not changes due to movement in the x-y space (the same path is followed for calculations both before and after training). Indeed, $|d\mathbf{x}\cdot\mathbf{D}'|$ must reduce during training for all x-y locations along the path. Theorem 6 requires that the magnitude of at least one of the components of $\mathbf{D}'$ be reduced. Choice of $d\mathbf{x}$ is simple in the case where both components decrease in magnitude; for this case $|\mathbf{D}'|$ becomes smaller and so will any dot product with a constant vector $d\mathbf{x}$. The situation is not as clear for the case where only one component decreases. Figure 47 illustrates two candidates for $d\mathbf{x}$ where one is directed in the x-direction and the other in the y-direction.

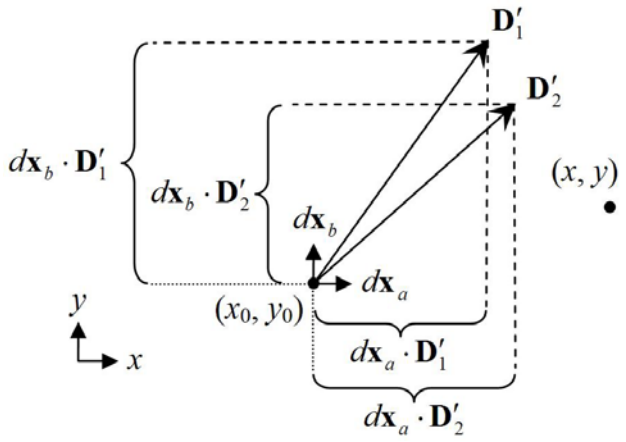


Figure 47. Two candidate directions for $d\mathbf{x}$ given a vector $\mathbf{D}'$ whose x-component increases in magnitude and whose y-component decreases.

Figure 47 arbitrarily indicates that the y-component of $\mathbf{D}'$ becomes smaller during training, but, because of the symmetry of the argument, choosing the x-component instead would produce the same result. The dot product in Figure 47 for the candidate $d\mathbf{x}$ in the x-direction will increase while the dot product will decrease for the

candidate in the *y*-direction. Due to the fact that the TAS is continuous, an angle between these two extremes must exist where $|d\mathbf{x}\cdot\mathbf{D}'|$ remains unchanged, as illustrated in Figure 48. Any direction between this angle and the vertical will produce a reduction in $|d\mathbf{x}\cdot\mathbf{D}'|$; a similar argument can be made using a $d\mathbf{x}$ pointing in the negative *y*-direction. Although the permissible directions may not point directly toward the desired point (x,y), a direction can be chosen which reduces the distance between (x_0,y_0) and (x,y). Appropriate choices for $d\mathbf{x}$ directions are made until (x,y) is eventually reached and a curve is defined where $|d\mathbf{x}\cdot\mathbf{D}'|$ is reduced at every point along it. It is possible that the direction chosen for $d\mathbf{x}$ could exceed either the *x* or *y* component of (x,y) at some location along the path, but that the component of the next choice for $d\mathbf{x}$ could then use the opposite sign and in such a way "tack" in on the correct point, as illustrated schematically in Figure 48.

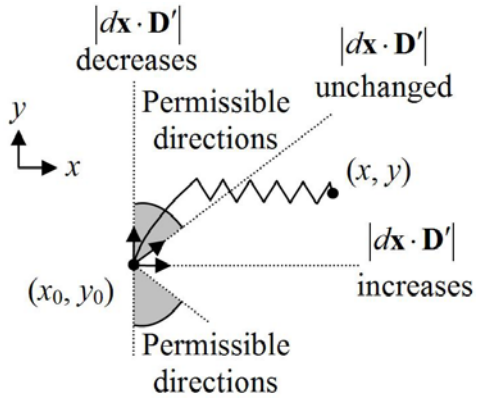


Figure 48. Illustration of permissible directions for $d\mathbf{x}$ for the case in Figure 47.

The jagged curve in Figure 48 represents an acceptable path from (x_0,y_0) to (x,y) for the case that $\mathbf{D}'$ is uniform in *x*-*y* space. In general, $\mathbf{D}'$ will of course vary in space, but Theorem 6 requires that at least one of the cardinal directions will be an acceptable

choice for $d\mathbf{x}$, and so an acceptable path will always exist. It is now proven that a path between any two points in the domain exists for which $|d\mathbf{x}\cdot\mathbf{D}'|$ is reduced during training at every location on the path given that $\left|D'_x(x,y)+D'_y(x,y)\right|$ is also reduced everywhere during training.

Theorem 8

Statement

Consider an ANN solving a DE as presented in the problem definition. The magnitude of the error in the TAS, $|D(x,y)|$, is reduced everywhere in the domain during any training iteration where $\left|D'_x(x,y)+D'_y(x,y)\right|$ is reduced everywhere in the domain.

Proof

Theorem 7 presented the quantity $d\mathbf{x}\cdot\mathbf{D}'$ as the error introduced into the approximate solution when moving a distance $d\mathbf{x}$ and proved that a path exists between any two points in the domain for which $|d\mathbf{x}\cdot\mathbf{D}'|$ decreases during training at all points along the path. The argument for the current theorem begins by choosing the starting point of the path in Theorem 7 on a Dirichlet boundary where the approximate solution is always exact. The error at any point within the domain is then computed by summing local values of $d\mathbf{x}\cdot\mathbf{D}'$ at every point along the path, starting on the boundary, and adding them to the zero error at the starting position. The error D at any particular point along the path depends recursively on the D value just "downstream" of the point in question as well as the local value of $d\mathbf{x}\cdot\mathbf{D}'$ (which must decrease in magnitude during training) as in (133). Equation (133) can be rewritten in the notation of Theorem 1 as

$$\left|D\left(\mathbf{x}+d\mathbf{x}\right)\right| = \left|a\left(\mathbf{x}\right)+b\left(\mathbf{x}\right)\right| \tag{142}$$

where a is the value of D evaluated at the location $\mathbf{x}$ and b is the additional error $d\mathbf{x}\cdot\mathbf{D}'$ introduced by the $d\mathbf{x}$ step. If $\mathbf{x}$ is chosen on the Dirichlet boundary, then $\left|D\left(\mathbf{x}+d\mathbf{x}\right)\right|$ must certainly decrease since a is always exactly zero and the magnitude of $b = d\mathbf{x}\cdot\mathbf{D}'$ decreases during training. When considering the following $d\mathbf{x}$ step along the path, the magnitudes of both quantities a and b are known to decrease during training. Theorem 1 can then be applied to ensure that the local value of $|D|$ for this step decreases as well. Theorem 1 is successively applied at every step along the path up to the point (x,y) to show that $|D|$ is reduced at every point along the path, and thus at every point in the domain since (x,y) is arbitrary.

Applying Theorem 1 requires comparing the relative sizes of a and b as well as the relative sizes of the arbitrarily small changes δa and δb introduced during training. The quantity a is the local error in the approximate solution, which is computed by summing the $b = d\mathbf{x}\cdot\mathbf{D}'$ values from the boundary up to the point in question as in (135) . As a sum, a will in general be larger than any single arbitrarily small component b of the sum. The only case where a is not obviously larger than b is for locations where a is small in size as well, i.e. with a near zero value which corresponds to locations in the domain where the approximate solution is exact, such as at Dirichlet boundaries.

Both the relative magnitudes and signs of a and b are important for consideration of Theorem 1. Both a and b will have the same sign for locations just off of the boundary since a is computed by adding b to zero. The magnitude of $b = d\mathbf{x}\cdot\mathbf{D}'$ may become smaller and eventually reach zero somewhere along the path, a point at which $\operatorname{sgn} a \neq \operatorname{sgn} b$. Such a sign change in b could come about if for example $d\mathbf{x}$ and $\mathbf{D}'$ were chosen to be mutually perpendicular or if $\mathbf{D}' = 0$ (which indicates that the trial and

analytical solutions have the same gradient at that point). The sign of $d\mathbf{x}\cdot\mathbf{D}'$ could also discontinuously change when tacking is required to approach the desired point (x,y), as illustrated by the jagged curve in Figure 48.

Understanding the possible combinations of relative sizes and signs for a and b is more easily understood by considering the quantities they represent. The quantity a is the local error in the approximate solution which is computed by summing the values of b from a Dirichlet boundary to the point in question. Figure 49 illustrates a path from a boundary point into the domain. The local error value is represented by vertical bars whose heights correspond to the signed summation of arrows up to the given point on the path. Figure 49 represents a snapshot in time of the values a and b along the path. A similar snapshot could be taken at a later time, after a training iteration, where each value of a and b have undergone small changes δa and δb; that is, the sizes of each bar and arrow have changed slightly. As with the quantities a and b themselves, the magnitude of δa at any point will in general be larger than δb since δa is the sum of the δb values up to that point. And again, the only exception being the case where a and b are on the same order of magnitude, i.e. when a is near zero in value.

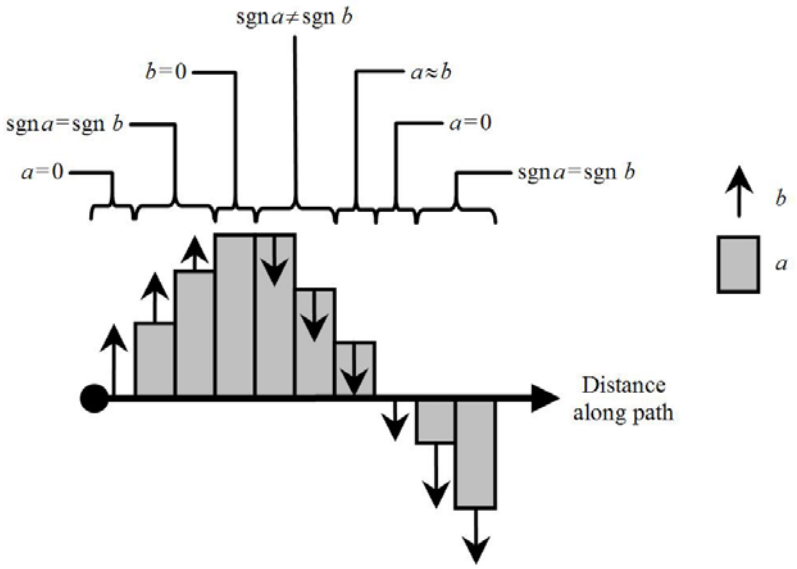


Figure 49. Illustration of possible relative sizes and signs of *a* (bars) and *b* (arrows) along a path as specified in Theorem 7 (note: not drawn to scale since arrows represent small changes and the bars represent accumulated changes).

The task now is to show, using Theorem 1, that the magnitude of $a+b$ reduces during training for all of the possible cases illustrated in Figure 49. Table 3 in Theorem 1 is especially helpful in identifying possible locations where the error might increase (note that the subscripts 1 and 2 in Table 3 indicate values before and after the training step, respectively). Locations where $\operatorname{sgn} a = \operatorname{sgn} b$ are immediately removed as possible problem locations since error must be reduced regardless of the relative sizes of *a* and *b*. Locations where either $a = 0$ or $b = 0$ do not present problems either since those quantities must remain zero (since they cannot be reduced) while the other quantity is reduced in magnitude. In general, $|a| > |b|$ and $|da| > |db|$ (regardless of the signs of *a* and *b*) and so the only possible problems for the $\operatorname{sgn} a \neq \operatorname{sgn} b$ case occur when *a* is near-zero in magnitude. At such locations, *a* and *b* will have equivalent magnitudes, making it irrelevant which quantity among δa and δb has a higher magnitude.

None of the requirements in Theorem 1 are fulfilled to result in an increase in $|a+b|$ for the various cases in Figure 49, and so the error $|D|$ must decrease everywhere along the path during training. Since such a path exists for all points in the domain according to Theorem 7, error must reduce at every location in the domain. Note that the error does not (and cannot!) reduce at locations where $a = D = 0$. At these locations, the error remains zero since it cannot reduce further.

Theorem 9

Statement

Consider an ANN solving a differential equation as presented in the problem definition. The quantity $|D'_x(x,y)+D'_y(x,y)|$ is reduced everywhere in the domain during any training iteration where the quantity $|D''_x(x,y)+D''_y(x,y)|$ is reduced everywhere in the domain.

Proof

The first partial derivatives of ψ for a differential change in position $d\mathbf{x} \equiv dx\hat{\mathbf{i}} + dy\hat{\mathbf{j}}$ may be expressed

$$\frac{\partial \psi(\mathbf{x}+d\mathbf{x})}{\partial x} = \frac{\partial \psi(\mathbf{x})}{\partial x} + dx\frac{\partial^2 \psi(\mathbf{x})}{\partial x^2} \text{ and } \frac{\partial \psi(\mathbf{x}+d\mathbf{x})}{\partial y} = \frac{\partial \psi(\mathbf{x})}{\partial y} + dy\frac{\partial^2 \psi(\mathbf{x})}{\partial y^2} \quad (143)$$

which may be combined to produce

$$\begin{aligned} \nabla\psi(\mathbf{x}+d\mathbf{x}) &= \frac{\partial \psi(\mathbf{x}+d\mathbf{x})}{\partial x}\hat{\mathbf{i}} + \frac{\partial \psi(\mathbf{x}+d\mathbf{x})}{\partial y}\hat{\mathbf{j}} \\ &= \left[\frac{\partial \psi(\mathbf{x})}{\partial x} + dx\frac{\partial^2 \psi(\mathbf{x})}{\partial x^2}\right]\hat{\mathbf{i}} + \left[\frac{\partial \psi(\mathbf{x})}{\partial y} + dy\frac{\partial^2 \psi(\mathbf{x})}{\partial y^2}\right]\hat{\mathbf{j}} \end{aligned} \quad (144)$$

Equation (143) and a similar definition for $\nabla\psi_t(\mathbf{x}+d\mathbf{x})$ are combined to produce

$$\nabla\psi_t(\mathbf{x}+d\mathbf{x})-\nabla\psi(\mathbf{x}+d\mathbf{x})=\left[\frac{\partial\psi_t(\mathbf{x})}{\partial x}-\frac{\partial\psi(\mathbf{x})}{\partial x}+dx\frac{\partial^2\psi_t(\mathbf{x})}{\partial x^2}-dx\frac{\partial^2\psi(\mathbf{x})}{\partial x^2}\right]\hat{\mathbf{i}}+\left[\frac{\partial\psi_t(\mathbf{x})}{\partial y}-\frac{\partial\psi(\mathbf{x})}{\partial y}+dy\frac{\partial^2\psi_t(\mathbf{x})}{\partial y^2}-dy\frac{\partial^2\psi(\mathbf{x})}{\partial y^2}\right]\hat{\mathbf{j}} \tag{145}$$

which is simplified using the notation

$$\begin{aligned}\mathbf{D}'(\mathbf{x}+d\mathbf{x})&=\left[D'_x(\mathbf{x})+dxD''_x(\mathbf{x})\right]\hat{\mathbf{i}}+\left[D'_y(\mathbf{x})+dyD''_y(\mathbf{x})\right]\hat{\mathbf{j}}\\&=D'_x(\mathbf{x}+d\mathbf{x})\hat{\mathbf{i}}+D'_y(\mathbf{x}+d\mathbf{x})\hat{\mathbf{j}}\end{aligned} \tag{146}$$

resulting in

$$\begin{aligned}\left|D'_x(\mathbf{x}+d\mathbf{x})+D'_y(\mathbf{x}+d\mathbf{x})\right|&=\left|D'_x(\mathbf{x})+dxD''_x(\mathbf{x})+D'_y(\mathbf{x})+dyD''(\mathbf{x})_y\right|\\&=\left|D'_x(\mathbf{x})+D'_y(\mathbf{x})+d\mathbf{x}\cdot\mathbf{D}''(\mathbf{x})\right|=\left|D'(\mathbf{x})+d\mathbf{x}\cdot\mathbf{D}''(\mathbf{x})\right|\end{aligned} \tag{147}$$

Theorem 8 can be used to prove this theorem by replacing $|D'|$ with $|D|$ and $|D''_x+D''_y|$ with $|D'_x+D'_y|$ as long as there exists a point (x_0,y_0) where $|D'_x+D'_y|$ is a known value which does not increase during the training iteration. According to the problem definition, the Dirichlet boundary condition is expressed as $\psi(s)$ where s is the location along the boundary. The gradient of the analytical solution at any s_0 on the boundary is then

$$\nabla\psi(s_0)=\left.\frac{d\psi}{ds}\frac{\partial s}{\partial x}\right|_{s=s_0}\hat{\mathbf{i}}+\left.\frac{d\psi}{ds}\frac{\partial s}{\partial y}\right|_{s=s_0}\hat{\mathbf{i}}=\frac{d\psi}{ds}\left[\frac{\partial s}{\partial x}\hat{\mathbf{i}}+\frac{\partial s}{\partial y}\hat{\mathbf{j}}\right]_{s=s_0} \tag{148}$$

The location of s_0 is chosen to correspond with the point (x_0,y_0), and the known value of $\nabla\psi_t(s_0)$ along with $\nabla\psi(s_0)$ from (148) is used in (124) to determine $\mathbf{D}'$. This finally leads to the determination of $|D'_x+D'_y|$ at the boundary. If s_0 is chosen to be a location on the Dirichlet boundary with a unity slope, then the value of $|D'_x+D'_y|$ must remain constant during training as developed in Theorem 6. Note that Theorem 6 requires existence of a point on a Dirichlet boundary which has a slope other than unity, whereas

the current theorem requires existence of a unity slope. Note that these slope requirements are placed on the local shape of the boundary, not on the derivative of the solution at that point. Both requirements can be fulfilled by rotating the domain as long as all Dirichlet boundaries do not have the same uniform slope. The base case in the recursive definition of (133) is now satisfied, clearing the way for development of analogous Theorems 7 and 8 for proving the current theorem.

Theorem 10

Statement

An ANN as described in the problem definition solving the nonhomogenous Laplace equation will reduce the error in the solution everywhere in the domain if the error in the differential equation is reduced everywhere in the domain during training.

Proof

The error function to be minimized for solving the nonhomogenous Laplace equation is (5) where

$$G = \nabla^2 \psi_t(\mathbf{x}) - f(\mathbf{x}) \tag{149}$$

The quantity $|G|$ is the magnitude of error in the DE, and in general decreases at most training points during each iteration of updating ANN parameters. Parameters such as the points in the training set T and the structure of the ANN are chosen so that the ANN has a good capability for generalization as described previously in this dissertation. Recall that a network with good generalization will reduce the error in the DE not only at the training points but everywhere in the domain. The magnitude of error in the DE is

$$|G| = \left|\nabla^2 \psi_t(\mathbf{x}) - f(\mathbf{x})\right| = \left|\nabla^2 \psi_t(\mathbf{x}) - f(\mathbf{x}) - \nabla^2 \psi(\mathbf{x}) + f(\mathbf{x})\right| = \left|D_x'' + D_y''\right| \tag{150}$$

since the differential equation is exactly satisfied for the analytical solution. An ANN with good generalization then will decrease $\left|D_x'' + D_y''\right|$ everywhere in the domain. Theorem 9 and then Theorem 8 are invoked to prove that the solution error will be reduced everywhere in the domain during any training iteration where the error in the DE is also reduced everywhere.

Theorem 11 – The Convergence Theorem

Statement

Consider the DE

$$\tilde{D}\psi(\mathbf{x}) = \sum_i f_i\left(\frac{\partial^{j_i}\psi}{\partial x_{k_i}^{j_i}}\right) = g(\mathbf{x}) \tag{151}$$

composed of i terms which are monotonic functions f_i of various j_i^{th} order partial derivatives with respect to the k_i^{th} component of the space vector $\mathbf{x}$, and $g(\mathbf{x})$ is an arbitrary function. The subscripts on f_i, j_i, and k_i indicate that each function in the summation may involve a partial derivative of different order and with respect to a different component of $\mathbf{x}$. The subscripts i are omitted for clarity when only one term in the summation is discussed. An ANN as described in the problem definition solving (151) will improve the error in the solution everywhere in the domain if the error in the DE is reduced everywhere in the domain during training.

Proof

Theorem 8 relates improvements in the first partial derivative to improvements in value. Theorem 9 extends Theorem 8 to treat second partial derivatives. Theorem 9 could in turn be extended further to higher order derivatives. It must be shown then that

$\left|D_x^{(j)} + D_y^{(j)}\right|$ decreases everywhere in the domain for some order derivative *j*. The need for this requirement was revealed when proving Theorem 10 for the Laplace equation.

It becomes apparent after further examination of Theorems 6 and 7 that a somewhat weaker requirement is sufficient. The "tacking" in Figure 48 can be accomplished as long as one term in $\left|D_x^{(j)} + D_y^{(j)}\right|$ decreases during training at every point in the domain. The requirement can be relaxed by observing that the progression of higher order derivatives considered in Theorem 8, Theorem 9, and any further extensions specify that each point in the domain must have at least one partial derivative whose error decreases in magnitude during training. Remember that the error in a partial derivative takes the form

$$D_k^{(j)} \equiv \frac{\partial^{(j)}\psi}{\partial x_k^{(j)}} - \frac{\partial^{(j)}\psi_t}{\partial x_k^{(j)}} \tag{152}$$

using the introduced notation. Ensuring then that every point in the domain has at least one decreasing $\left|D_k^{(j)}\right|$ of any order *j* and component *k* is sufficient to ensure a reduction in the error of the solution.

The error in the DE for the given TAS is

$$|G| = \left|\sum_i f_i\left(\frac{\partial^{j_i}\psi_t}{\partial x_{k_i}^{j_i}}\right) - g(\mathbf{x})\right| \tag{153}$$

which can be combined with (151) to generate

$$\begin{aligned} |G| &= \left|\sum_i f_i\left(\frac{\partial^{j_i}\psi_t}{\partial x_{k_i}^{j_i}}\right) - g(\mathbf{x}) - \sum_i f_i\left(\frac{\partial^{j_i}\psi}{\partial x_{k_i}^{j_i}}\right) + g(\mathbf{x})\right| \\ &= \left|\sum_i \left[f_i\left(\frac{\partial^{j_i}\psi_t}{\partial x_{k_i}^{j_i}}\right) - f_i\left(\frac{\partial^{j_i}\psi}{\partial x_{k_i}^{j_i}}\right)\right]\right| \end{aligned} \tag{154}$$

If $|G|$ decreases at all points in the domain, then the magnitude of at least one term in the summation in (154) must also decrease. Since the functions f_i are monotonic, then at least one $\left|D_k^{(j)}\right|$ must decrease for all points in the domain, thus proving this theorem.

Linear and exponential functions are examples of permissible forms for f_i in (151). This theorem can in fact be applied even to DEs composed of non-monotonic functions as long as the trial and true solutions are sufficiently close so that the range

$$\left[f\left(\frac{\partial^j \psi}{\partial x_k^j}\right), f\left(\frac{\partial^j \psi_t}{\partial x_k^j}\right)\right] \tag{155}$$

is monotonic. Figure 50 illustrates an example of a non-monotonic function for which the range in question is monotonic. The requirement that the range in (155) be monotonic would likely be fulfilled for any TAS of even modest accuracy, thus extending the application of this theorem to essentially any DE.

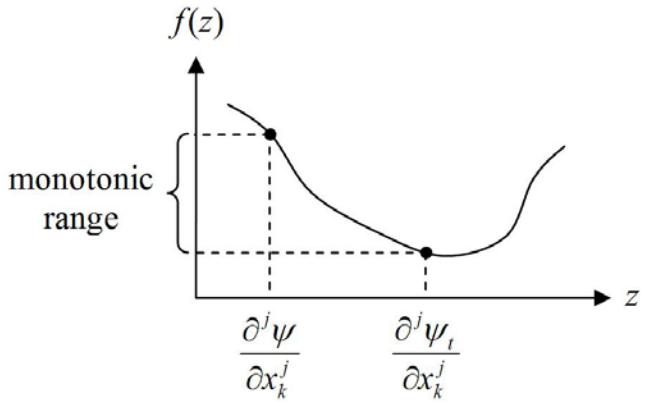


Figure 50. Example of a non-monotonic function for which the trial and true solutions are sufficiently close so that Theorem 11 can still be applied.

References

[1] L. Shu-Hsien, "Expert system methodologies and applications - a decade review from 1995 to 2004," *Expert Systems with Applications*, vol. 28, 2005.

[2] I. Turksen, "Fuzzy logic: review of recent concerns," presented at IEEE International Conference on Systems, Man, and Cybernetics, 1997.

[3] M. Meireles, P. Almeida, and M. Simoes, "A comprehensive review for industrial applicability of artificial neural networks," *IEEE Transactions on Industrial Electronics*, vol. 50, pp. 585-601, 2003.

[4] B. Dolin and J. Merelo, "Resource review: a web-based tour of genetic programming," *Genetic Programming and Evolvable Machines*, vol. 3, pp. 311-313, 2002.

[5] M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: a review," *SIGMOD Record*, vol. 34, pp. 18-26, 2005.

[6] A. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, pp. 433-460, 1950.

[7] W. Bushor, "The perceptron - an experiment in learning," *Electronics*, vol. 33, pp. 56-59, 1960.

[8] S. Khanna and C. Noback, "Neural nets and artificial intelligence," *Artificial Intelligence*, pp. 83-88, 1963.

[9] V. Kuznetsova, V. Kuzmenko, and I. Tsygelnyi, "Problems and future trends in neuron engineering," *Otbor i Peredacha Informatsii*, vol. 62, pp. 49-55, 1980.

[10] T. Maxwell, C. Giles, Y. Lee, and H. Chen, "Nonlinear dynamics of artificial neural systems," presented at Neural Networks for Computing, Snowbird, UT, USA, 1986.

[11] A. Tsoi, "Multilayer perceptron trained using radial basis functions," *Electronic Letters*, vol. 25, pp. 1286-1297, 1989.

[12] E. Grant and B. Zhang, "A neural-net approach to supervised learning of pole balancing," presented at IEEE International Symposium on Intelligent Control, Albany, NY, USA, 1989.

[13] A. Barto and R. Sutton, "Landmark learning: an illustration of associative search," *Biological Cybernetics*, vol. 42, pp. 1-8, 1981.

[14] K. Hornik, M. Stichcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359-365, 1989.

[15] L. Prechelt, "A quantitative study of experimental evaluations of neural network learning algorithms: current research practice," *Neural Networks*, vol. 9, 1996.

[16] D. Papageorgiou, I. Demetropoulos, and I. Lagaris, "Merlin-3.0, a multidimensional optimization environment," *Computational Physics Communications*, vol. 109, pp. 227-249, 1998.

[17] Z. Yong-Jian and B. Bhanu, "Adaptive object detection from multisensor data," presented at International Conference on Multisensor Fusion and Integration for Intelligent Systems, Washington DC, USA, 1996.

[18] N. Wanas and M. Kamel, "Decision Fusion in neural network ensembles," presented at International Joint Conference on Neural Networks, Washington DC, USA, 2001.

[19] P. Palmes, T. Hayasaka, and S. Usui, "Evolution and adaptation of neural networks," presented at International Joint Conference on Neural Networks, Portland, Oregon, 2003.

[20] T. Leephakpreeda, "Novel determination of differential-equation solutions: universal approximation method," *Journal of Computational and Applied Mathematics*, vol. 146, pp. 261-271, 2002.

[21] I. Lagaris, A. Likas, and D. Papageorgiou, "Neural-Network Methods for Boundary Value Problems with Irregular Boundaries," *IEEE Transactions on Neural Networks*, vol. 11, pp. 1041-1049, 2000.

[22] N. Mai-Duy and T. Tran.Cong, "Numerical solution of differential equations using multiquadric radial basis function networks," *Neural Networks*, vol. 14, pp. 185-199, 2001.

[23] L. Jianyu, L. Siwei, Q. Yingjian, and H. Yaping, "Numerical Solution of Elliptic Partial Differential Equation by Growing Radial Basis Function Neural Networks," presented at International Joint Conference on Neural Networks, Hawaii, USA, 2002.

[24] G. Smith, *Numerical solution of partial differential equations: finite difference methods*. Oxford: Clarendon Press, 1978.

[25] T. Hughes, *The finite element method*. New Jersey: Prentice-Hall, 1987.

[26] C. Brebbia, J. Telles, and L. Wrobel, *Boundary element techniques: theory and applications in engineering*. Berlin: Springer-Verlag, 1984.

[27] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations," *IEEE Transactions on Neural Networks*, vol. 9, pp. 987-1000, 1998.

[28] K. McFall and J. R. Mahan, "Investigation of weight reuse in multi-layer perceptron networks for accelerating the solution of differential equations," presented at International Conference on Intelligent Systems Design and Applications, Budapest, Hungary, 2004.

[29] K. McFall and J. R. Mahan, "Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions," *IEEE Transactions on Neural Networks*, under review.

[30] M. Reidmiller and H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," presented at International Conference on Neural Networks, San Francisco, California, 1993.

[31] V. Kecman, *Learning and Soft Computing*. Cambridge: The MIT Press, 2001.

[32] M. Hüsken and C. Goerick, "Fast learning for problem classes using knowledge based network initialization," presented at IEEE-INNS-ENNS International Conference on Neural Networks, 2000.

[33] D. Parisi, M. Mariani, and M. Laborde, "Solving Differential Equations with Unsupervised Neural Networks," *Chemical Engineering and Processing*, vol. 42, pp. 715-721, 2003.

[34] M. Hüsken, C. Goerick, and A. Vogel, "Fast Adaptation of the Solution of Differential Equations to Changing Constraints," presented at 2nd International ICSC Symposium on Neural Computation, Berlin, Germany, 2000.

[35] W. Gordon and C. Hall, "Exact matching of boundary conditions and incorporation of semiquantitative solution characteristics in initial approximations to boundary value problems," *Journal of Computational Physics*, vol. 25, pp. 151-162, 1977.

[36] W. Gordon, "Blending function methods of bivariate and multivariate interpolation and approximation," *SIAM Journal on Numerical Analysis*, vol. 8, pp. 158-177, 1971.

[37] J. Faires and R. Burden, *Numerical Methods*: Thomson Brooks/Cole, 1998.

[38] J. Jang, C. Sun, and E. Mitzutani, *Neuro-Fuzzy and Soft Computing*. New Jersey: Prentice Hall, 1997.

[39] J. Jang and C. Sun, "Neuro-fuzzy modeling and control," *Proceedings of the IEEE*, vol. 83, pp. 378-406, 1995.

[40] H. Lee and I. Kang, "Neural Algorithms for Solving Differential Equations," *Journal of Computational Physics*, vol. 91, pp. 110-117, 1990.

[41] A. Meade and A. Fernandez, "The Numerical Solution of Linear Ordinary Differential Equations by Feedforward Neural Networks," *Mathematical Computational Modelling*, vol. 19, pp. 1-25, 1994.

[42] R. Yentis and M. Zaghoul, "VLSI Implementation of Locally Connected Neural Network for Solving Partial Differential Equations," *IEEE Transactions on Circuits and Systems I*, vol. 43, pp. 687-690, 1996.

[43] S. He, K. Reif, and R. Unbehauen, "Multilayer neural networks for solving a class of partial differential equations," *Neural Networks*, vol. 13, pp. 385-396, 2000.

[44] J. Yam and T. Chow, "Feedforward networks training speed enhancement by optimal initialization of the synaptic coefficients," *IEEE Transactions on Neural Networks*, vol. 12, pp. 430-434, 2001.

[45] S. Raudy, "Prior weights in adaptive pattern recognition," presented at 15th International Conference on Pattern Recognition, 2000.

[46] W. Gordon and C. Hall, "Construction of curvilinear coordinate systems and applications to mesh generation," *International Journal for Numerical Methods in Engineering*, vol. 7, pp. 461-477, 1973.

[47] W. Gordon and C. Hall, "Transfinite element methods: Blending function interpolation over arbitrary curved element domains," *Numerical Mathematics*, vol. 21, pp. 109-129, 1973.

[48] H. Kardestunger, *Finite Element Handbook*. New York: McGraw-Hill, 1987.

[49] K. Grandek, "Convergence of a hybrid iterative procedure," *Mathematics and Computers in Simulation*, vol. 22, pp. 319-323, 1980.

[50] T. Jankowski, "Multistep methods for nonlinear boundary-value problems with parameters," *Journal of Mathematical Analysis and Applications*, vol. 147, pp. 1-11, 1990.

[51] H. Schmitt, "The domain of convergence for the iterative solution of nonlinear second order boundary value problems," *Archive for Rational Mechanics and Analysis*, vol. 93, pp. 301-322, 1986.

[52] R. Agarwal, "Computational methods for discrete boundary value problems, II," *Journal of Mathematical Analysis and Applications*, vol. 166, pp. 540-562, 1992.

[53] R. Agarwal and R. Usmani, "Monotone convergence of iterative methods for right focal point boundary value problems," *Journal of Mathematical Analysis and Applications*, vol. 130, pp. 451-459, 1988.

[54] D. Braess, *Finite Elements*. Cambridge: Cambirdge University Press, 1997.

[55] M. Adams, "Multigrid Equation Solvers for Large Scale Nonlinear Finite Element Simulations," in *PhD dissertation in Civil Engineering*: University of California Berkeley, 1998.