Inversion of Feedforward Neural Networks: Algorithms and Applications

CRAIG A. JENSEN, RUSSELL D. REED, ROBERT J. MARKS, II, FELLOW, IEEE, MOHAMED A. EL-SHARKAWI, FELLOW, IEEE, JAE-BYUNG JUNG, ROBERT T. MIYAMOTO, GREGORY M. ANDERSON, AND CHRISTIAN J. EGGEN

Invited Paper

Feedforward layered perceptron neural networks seek to capture a system mapping inferred by training data. A properly trained neural network is not only capable of mimicking the process responsible for generating the training data, but the inverse process as well. Neural network inversion procedures seek to find one or more input values that produce a desired output response for a fixed set of synaptic weights. There are many methods for performing neural network inversion. Multi-element evolutionary inversion procedures are capable of finding numerous inversion points simultaneously. Constrained neural network inversion requires that the inversion solution belong to one or more specified constraint sets. In many cases, iterating between the neural network inversion solution and the constraint set can successfully solve constrained inversion problems. This paper surveys existing methodologies for neural network inversion, which is illustrated by its use as a tool in query-based learning, sonar performance analysis, power system security assessment, control, and generation of codebook vectors.

Keywords*—*Adaptive sonar, constrainted inversion, feedforward neural networks, multilayer perceptron, nonlinear system inversion, power system security assessment, query-based learning.

I. INTRODUCTION

For a given set of training data and through proper training, a feedforward layered perceptron neural network is ideally able to synthesize a mapping akin to the process that is responsible for generating the training data. The performance of a trained feedforward perceptron neural network can be characterized by

$$o_k = f_k(\mathbf{i}, \mathbf{w}) \tag{1}$$

where o_k is the kth neural network output corresponding to a vector input vector $\mathbf{i}$, $\mathbf{w}$ is a vector of the weights internal to the network, and $f_k(\cdot)$ is a memoryless function describing the mapping from the input to the kth output. The structure of the feedforward perceptron (i.e., the number of hidden neurons and the neuron connections, or weights) is imbedded in $f_k(\cdot)$. The neural network is trained by fixing the input and output and adjusting the weights until an acceptable performance is achieved. If a single scalar output is assumed, o_k can be replaced by o and $f_k(\cdot)$ by $f(\cdot)$ in (1).

Inversion of a neural network consists of clamping the weights and the neural network output while adjusting the input in (1) until an equality or a best possible fit occurs for one or more values of $\mathbf{i}$. In the analysis to follow, the weights of the neural network are assumed fixed. The dependence of the output on $\mathbf{w}$ will therefore be assumed implicitly in the notation $f(\mathbf{i}) = f(\mathbf{i}, \mathbf{w})$.

In general, as illustrated in Fig. 1 for two inputs, i_1 and i_2, ($\mathbf{i} = [i_1\, i_2]^T$) and one output, $f(\mathbf{i})$, numerous different inputs can generate the same output. Each contour of the plot in Fig. 1 corresponds to $f(\mathbf{i}) = c$ for a different constant c. Inversion is generally not unique when the input dimension is greater than the output dimension. It consists, rather, for a given c, of finding one or more elements of the set of inputs Λ on a contour (or a set of disjoint contours) where

$$\Lambda = \{\mathbf{i} : f(\mathbf{i}) = c\}. \tag{2}$$

Depending on the application, feedforward neural network inversion focuses on finding: 1) any solution point in Λ; 2) a point or points in Λ obeying one or more externally imposed constraints; or 3) a number of evenly dispersed points in Λ.

II. INVERSION TECHNIQUES

A common formulation of the inversion problem is to establish an objective function, e.g.,

$$E(\mathbf{i}) = (f(\mathbf{i}) - c)^2 \tag{3}$$

Manuscript received March 30, 1998; revised May 7, 1999. This work was supported by the Office of Naval Research (ONR), the National Science Foundation (NSF), and the Electric Power Research Institute (EPRI).

C. A. Jensen, R. D. Reed, R. J. Marks, II, M. A. El-Sharkawi, and J.-B. Jung are with the Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA.

R. T. Miyamoto, G. M. Anderson, and C. J. Eggen are with the Applied Physics Laboratory, University of Washington, Seattle, WA 98105 USA.

Publisher Item Identifier S 0018-9219(99)06911-X.

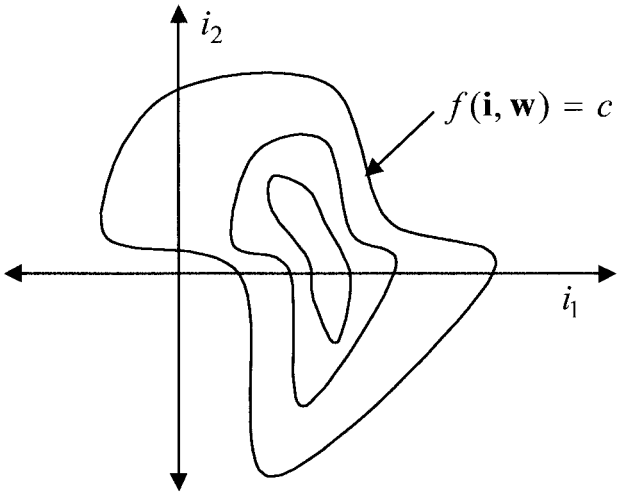


Fig. 1. The inversion of a neural network typically has numerous solutions. Each of the input ordered pairs lying on a given contour generate the same neural network output. The contours can be disjoint.

and then to search for Λ, or a subset thereof, that minimizes the error for a specified c. The number of points needed from the set Λ is application dependent. Neural network inversion algorithms can be placed into three broad classes:

1) exhaustive search;
2) single-element search methods [2]–[4];
3) multicomponent (i.e., population-based) evolutionary methods that operate on a plurality of potential solutions [5], [6].

In choosing among inversion techniques, exhaustive search[1] should be considered when both the dimensionality of the input and allowable range of each input variable is low. The simplicity of the approach coupled with the swiftness in which a layered perceptron can be executed in the feedforward mode make this approach ever more attractive as computational speed increases. In single-element search, one search point explores the landscape E defined for fixed $\mathbf{w}$ as a function of $\mathbf{i}$. An example is a gradient-based method similar to the gradient descent error backpropagation learning algorithm [2]. Only one point on Λ is found for each search and is typically dependent on the initialization. Multicomponent evolutionary methods, on the other hand, seek to minimize the objective function using numerous search points in turn resulting in numerous solutions on or close to Λ.

A. Single-Element Search

1) Williams, Linden, and Kinderman (WLK) Inversion: The notion of single-element network inversion via the gradient approach was first proposed by Williams [2] and later by Linden and Kinderman [3]. These methods make use of standard error backpropagation optimization. The search is initialized with an input vector $\mathbf{i}^0$. If i_k^t is the kth component of the vector $\mathbf{i}^t$, then gradient descent suggests the recursion

$$i_k^{t+1} = i_k^t - \eta \frac{\partial E}{\partial i_k^t} \tag{4}$$

where η is the step size and t is the iteration index. Familiar alterations can be made to the recursion (e.g., use of momentum and scheduling step sizes) to improve both the rate of convergence and the final error level [7].

Assuming a general feedforward topology, the iteration for inversion in (4) can be solved as follows, $\partial E/\partial i_k = \delta_k$, $k \in I$ where, for any neuron

$$\delta_j = \begin{cases} \varphi_j'(o_j)(o_j - t_j):, & j \in O \\ \varphi_j'(o_j) \displaystyle\sum_{m \in H, O} \delta_j w_{jm}:, & j \in I, H \end{cases} \tag{5}$$

and

I, H, O the sets of input, hidden, and output neurons, respectively;
w_{jm} the weight value from neuron j to neuron m;
φ_j' the derivative of the jth neuron squashing function;
o_j the activation of the jth neuron;
t_j the desired output of the jth neuron.

Note the neuron derivatives, δ_j, in (5) must be solved in a backward order from output to input similar to the standard backpropagation algorithm. The absence of feedback is the only assumption made regarding the neuron connections in (5).

Example 1—Sonar Performance Analysis: A useful application of WLK inversion is analysis of sonar performance under various environmental conditions. The scenario is shown in Fig. 2. A surface ship controls the depth to which a sonar unit is submerged. The surveillance area assigned to the sonar is shown in Fig. 2 as a shaded region. For analysis, the surveillance area is divided into pixels. For each pixel, the signal to interference ratio (SIR) delivered by the sonar can be computed using computationally intensive software emulating acoustic propagation. The SIR at each pixel is a function of a number of parameters including sonar depth, wind speed (surface roughness), bottom type, and sound velocity as a function of depth. An identified target area, shown by a black square in Fig. 2, can consist of one or more pixels. The inverse problem is to determine a set or subset of input parameters that will yield a high SIR in the target area.

A neural network approach to this problem is illustrated in Fig. 3. A neural network is trained to generate SIR pixel values as a function of sonar and environmental parameters. Once trained, the inverted neural network can provide input parameters to generate desired SIR performance in a specified target region. In Fig. 3, the sonar parameters are assumed fixed and are therefore clamped to specific values. The target portion of the surveillance region specifies a subset of output pixels. The remaining pixels in the surveillance region are in a "don't care" category and are allowed to float in the inversion process to arbitrary values typically constrained to lie within a specified range. WLK

[1] When the neural network input space is R_n, exhaustive search is not possible. In such cases, the input space is typically sampled at a predefined interval over which exhaustive search can be performed.

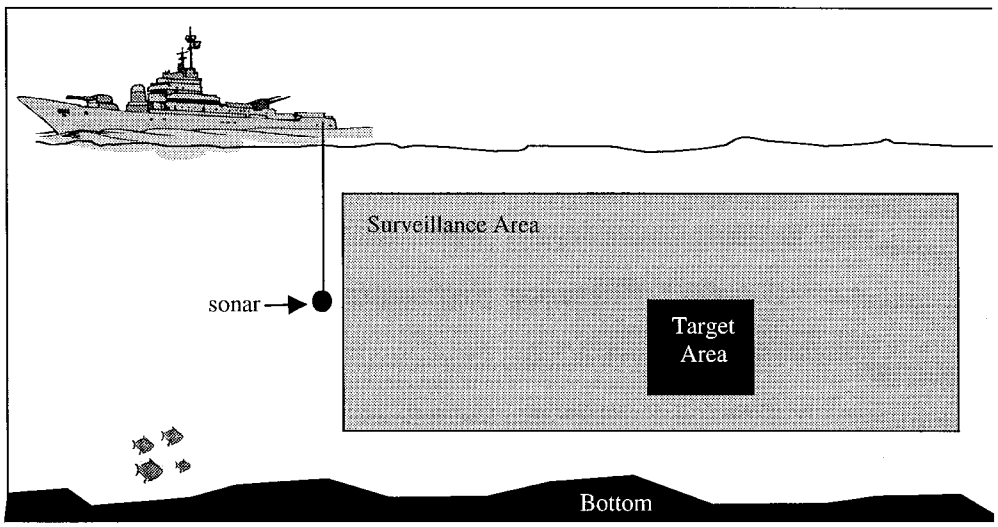


Fig. 2. An illustration of the sonar performance analysis problem. A neural network is trained to evaluate the signal to interference ratio (SIR) at each pixel in the surveillance area as a function of environmental and sonar parameters.

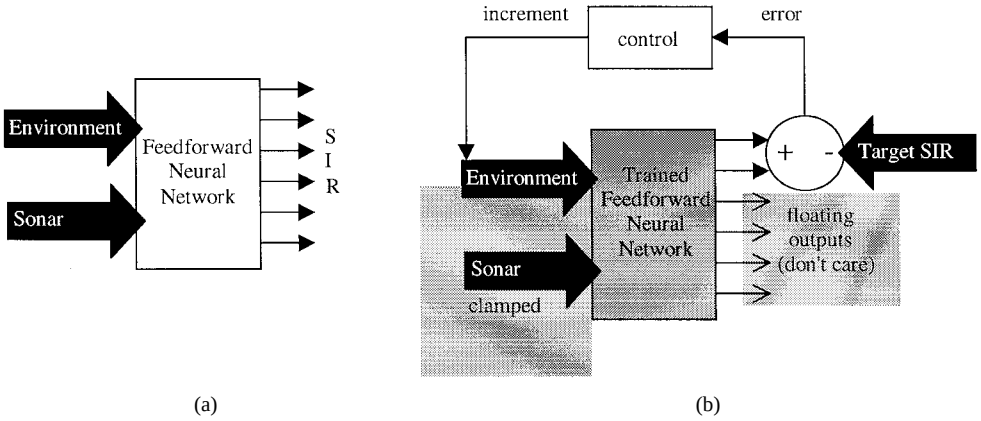


Fig. 3. (a) A feedforward neural network is trained on data generated by computationally intensive emulation software. The output SIR is a function of both environmental and sonar parameters. (b) For fixed SIR's in a specified target region and for designated sonar parameters, the neural network is being inverted to determine corresponding environmental parameters. The neural network output corresponding to pixels not contained in the target region are allowed to float, i.e., take on arbitrary values.

inversion of the neural net in Fig. 3 is performed on the environmental input parameters. The result indicates an environment in which the clamped sonar parameters work best. Alternately, the net can be inverted to specify the best sonar settings for a fixed set of environmental parameters. In this case, only a single point on Λ is needed because any solution that maximizes the sonar performance in the target region is acceptable.

In practice, for a given target region, the exact SIR values to invert are unknown. Rather, the maximum deliverable SIR is desired. This can be achieved by clamping the neural network pixel outputs in the target region to a value that exceeds the maximum achievable SIR ratio as determined by the training data. In such a case, the objective function $E(\mathbf{i})$ cannot reach zero. The minimum value reached by the error function, however, corresponds to selected values of $\mathbf{i}$ that produce the maximum SIR in the target region.

Using data generated by the sonar simulation software, a neural network consisting of five inputs, three hidden layers of 15, 10, and 30 neurons, respectively, and 390 outputs were trained. Five inputs are supplied, of which four are environmental parameters (wind speed, bottom type, and two samples of sound speed at different depths) and one is the sonar depth. There are 390 outputs, corresponding to a grid of 13 depth pixels by 30 range pixels covering, respectively, a surveillance range of 180 m by 6 km. During training, the data input for each pixel is normalized to give a value of one for the maximum training pixel value and a zero for the minimum. For inversion, the depth input and bottom type (sand) were clamped and the three remaining environmental parameters were evaluated from the inversions. The surveillance area was tiled with 2×2 nonoverlapping target pixel regions (except the final row because the 13 pixel depth is not divisible by 2). Inversions were performed sequentially for each 2×2 pixel target. Each target pixel was clamped to a value of 1.0, corresponding to the normalized maximum achievable value. Plots of the inverted SIR and absolute error between

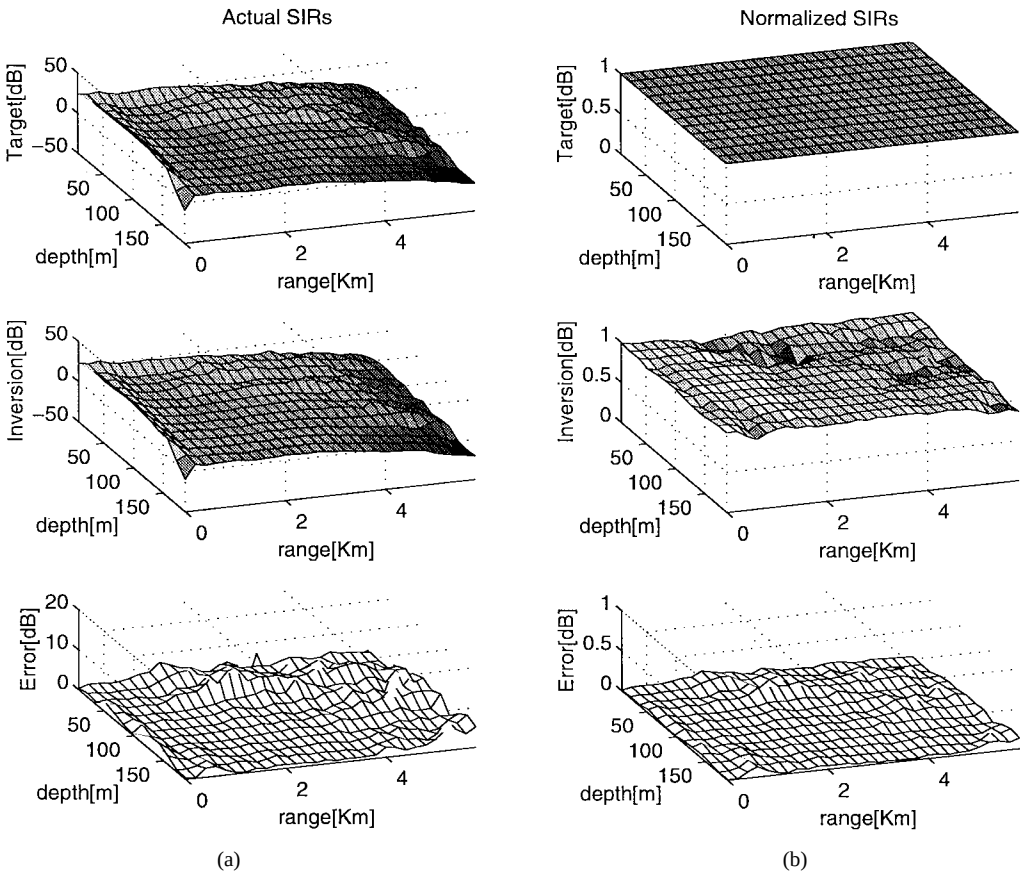


Fig. 4. Inversion performance of a neural network trained to calculate the acoustic SIR as a function of environmental and sonar parameters. The inversion is performed on a neural network trained such that all training data are normalized to fit inside a unit box, i.e., the pixel data were normalized to one for the maximum pixel value input and zero for the minimum. The inversion for the unnormalized training and the resulting error are shown in (a). The corresponding normalized plots are shown in (b).

the model-generated target and the inverted SIR are shown in Fig. 4.

An alternate approach to this problem is replacement of the trained feedforward neural network in Fig. 3 with the Applied Physics Laboratory (APL) sonar simulation. The software's slow execution time, however, renders this approach ineffective. The speed at which neural network inversion is performed and, as is illustrated in Fig. 4, the observed accuracy are distinct advantages of neural inversion approach to this problem.

2) Nearest Inversion—Finding the Inverse Solution Lying Nearest to a Specified Point: Nearest inversion [22] is a single-element search method that addresses the following problem. Given a function $f(\mathbf{i})$, a target output level c, and an initial base point $\mathbf{i}_0$, find the point $\mathbf{i}^*$ that satisfies $f(\mathbf{i}^*) = c$ and is closest to $\mathbf{i}_0$ in some sense. The Euclidean distance is used in the analysis and applications to follow. In applications, the base point $\mathbf{i}_0$ might represent the operating point of a controlled system and the solution $\mathbf{i}^*$ could be the nearest point achieving a desired operating performance or a danger point to be avoided. An example application of nearest inversion to power security assessment is offered in Section III.

Nearest inversion is a constrained optimization problem. One method to find a solution is outlined below. To simplify the description, define "inside" and "outside" as $f(\mathbf{i}) > c$ inside the surface and $f(\mathbf{i}) < c$ outside. Assume the base point $\mathbf{i}_0$ is inside the surface, $f(\mathbf{i}_0) > c$. If it is not, use the function $c - f(\mathbf{i})$ instead.

The basic idea is to generate random points $\mathbf{i}$ around $\mathbf{i}_0$ and:

1) for points outside the surface, use interval halving or a similar scheme to locate the surface;
2) for points inside the surface, $f(\mathbf{i}) > c$, follow the gradient down to the surface;
3) once the points reach the surface, perform constrained gradient descent to minimize the distance $||\mathbf{i}_0 - \mathbf{i}||$, i.e., move $\mathbf{i}$ toward $\mathbf{i}_0$ while staying on the surface.

The initial points sample a region of interest that contains both $\mathbf{i}_0$ and some part of the boundary. Steps 1) and 2) move the candidate points onto the surface and step 3) then moves the points along the surface toward $\mathbf{i}_0$.

a) Generation of initial candidates: Initial search points are generated by adding Gaussian noise to $\mathbf{i}_0$ with the variance σ^2 adjusted dynamically so that approximately half of the initial points fall on either side of the boundary. The goal is to produce points both near $\mathbf{i}_0$ and near the boundary surface. By requiring that a significant fraction of the initial

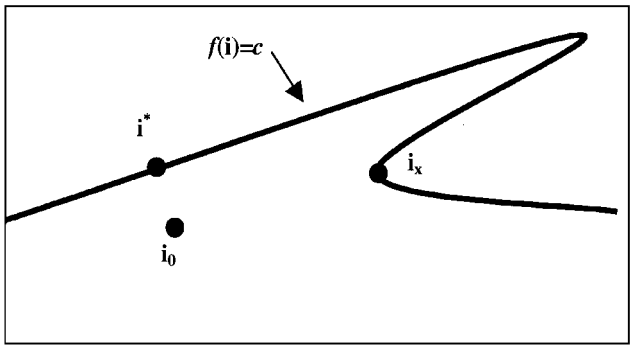


Fig. 5. Nearest inversion seeks to find the point $\mathbf{i}^*$ on the manifold $f(\mathbf{i}) = c$ closest to a base point $\mathbf{i}_0$. In this figure, the point $\mathbf{i}_x$ is a local minimum solution to be avoided.

points lie on either side of the boundary, we ensure that the search diameter is at least as large as the distance from $\mathbf{i}_0$ to the boundary. If enough initial candidates are generated in this region, there is a reasonable chance of avoiding potential local minima such as is illustrated in Fig. 5. When $\mathbf{i}_0$ is near the surface, σ^2 will be small and remote points in irrelevant regions will be ignored. Similarly, when $\mathbf{i}_0$ is far from the surface, σ^2 will be larger and a larger area will be searched.

Ideally, a significant fraction of the initial points should lie on both sides of the surface. This, however, may not be attainable because, with curved surfaces in high-dimensional spaces, a spherical noise distribution is unlikely to yield equal numbers of points on both sides of the boundary. Sometimes most of the input space is either inside or outside the surface. In such cases, candidates on the same side as $\mathbf{i}_0$ can be found by making σ^2 sufficiently small. Finding points on opposite sides may be difficult, however, unless σ^2 is much larger than the distance from $\mathbf{i}_0$ to the surface.

A running tally is kept of the number of points on either side of the surface and σ^2 is adjusted as new candidates are generated. Since the input is usually normalized, σ^2 can be initialized to 0.1. If none of the candidates lies on the opposite side, the initial search scale is too small and none of the candidates is near the surface, i.e., σ^2 is too small and should be increased. If most candidates fall on the opposite side, then σ^2 is too large and should be reduced.

The number of required candidate points N depends on the dimension of the space and the complexity of the boundary surface. More points improve the probability of finding the true nearest point at the expense of increasing computation time. If N is too small, the algorithm does not see enough of the function and may return a poor result if the surface has local minima. The search time might also be unnecessarily slow in the constrained gradient following phase.

b) Moving the points to the surface: The next step is to move candidate points to the boundary contour. There are two cases depending on if $\mathbf{i}$ lies inside or outside the surface.

1) *Points Inside the Surface:* If candidate point $\mathbf{i}$ is inside the surface ($f(\mathbf{i}) > c$), then follow the negative gradient down to the boundary. In many cases, a simple gradient descent procedure with an adaptive step size is adequate. If $f(\mathbf{i})$ is smooth and local minima are not a problem, more efficient methods may be used. If the gradient exists but is not available, less efficient evaluation-only gradient-following methods can be used.
2) *Points Outside the Surface:* If the candidate point $\mathbf{i}$ is outside the surface, the boundary contour must lie between $\mathbf{i}$ and $\mathbf{i}_0$ and the problem is one of finding a zero of $f(\mathbf{i}) - c$ along the line from $\mathbf{i}_0$ to $\mathbf{i}$. There are standard routines for this which can be very efficient if $f(\mathbf{i})$ is smooth. Our implementation uses interval-halving methods as a compromise between robustness and efficiency. Although first- and second-order interpolation methods may be faster when $f(\mathbf{i})$ is approximately linear or quadratic between $\mathbf{i}$ and $\mathbf{i}_0$, they are inefficient when $f(\mathbf{i})$ approaches a step function as is common in neural network classifiers.

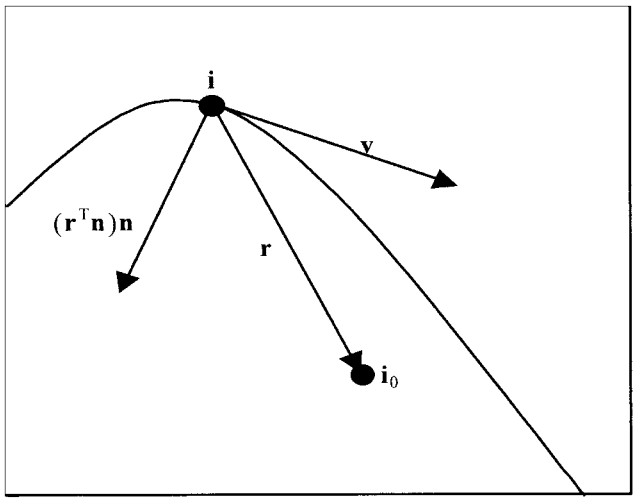


Fig. 6. The tangent vector. The vector $\mathbf{v} = \mathbf{r} - (\mathbf{r}^T\mathbf{n})\mathbf{n}$ is the component of the vector $\mathbf{i}_0 - \mathbf{i}$ tangent to the surface $\mathbf{i}$. Infinitesimal movement along $\mathbf{v}$ minimizes the distance $\mathbf{i}_0 - \mathbf{i}$ subject to the constraint $f(\mathbf{i}) = c$.

c) Minimizing the distance to $\mathbf{i}_0$*:* Once the candidate point is effectively on the surface, $f(\mathbf{i}) - c < \varepsilon$ where ε is a tolerance parameter, the next step is to move it along the surface to reduce the distance $|\mathbf{i} - \mathbf{i}_0|$. This is a constrained minimization problem: minimize $E = |\mathbf{i} - \mathbf{i}_0|$ subject to $f(\mathbf{i}) = c$.

A gradient descent method to minimize this is described below (see Fig. 6). Let n be the unit vector normal to the boundary surface, i.e., the gradient of f with respect to $\mathbf{x}$, normalized to unit length. The vector $\mathbf{r} = \mathbf{i}_0 - \mathbf{i}$ points from $\mathbf{i}$ to $\mathbf{i}_0$ and has a component of length $\mathbf{r}^T\mathbf{n}$ normal to the surface. Removal of this normal component yields the tangent vector $\mathbf{v} = \mathbf{r} - (\mathbf{r}^T\mathbf{n})\mathbf{n}$. When $\mathbf{i}$ is at a local minimum of the distance function, $\mathbf{v}$ vanishes because $\mathbf{r}$ is parallel to $\mathbf{n}$.

Ideally, infinitesimal movement along $\mathbf{v}$ reduces the distance to $\mathbf{i}_0$ while staying on the boundary surface. For

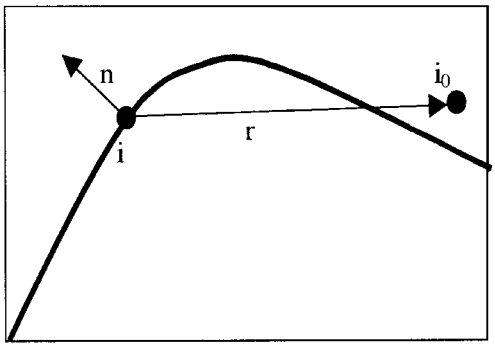


Fig. 7. If $\mathbf{v}^T\mathbf{n} < 0$, the surface normal (into the surface) points in a direction opposed to the vector from $\mathbf{i}$ to $\mathbf{i}_0$. In this case, $\mathbf{i}$ is considered to be outside the surface and a zero-finding step is performed to move across the "valley" to the section of the surface nearer to $\mathbf{i}_0$.

noninfinitesimal step sizes and highly curved surfaces, a step along $\mathbf{v}$ may fall out of the tolerance region, $|f(\mathbf{i}) - c| < \varepsilon$, and it may be necessary to move the point back onto the boundary.

d) Other cases: Occasionally the case illustrated in Fig. 7 may arise. If $\mathbf{v}^T\mathbf{n} < 0$, the surface normal (pointing "into" the surface) points in a direction opposed to the vector from $\mathbf{i}$ to $\mathbf{i}_0$. In this case, $\mathbf{i}$ is considered to be outside the surface and a zero-finding step is done to move across the "valley" to the section of the surface nearer to $\mathbf{i}_0$.

B. Evolutionary Methods

Multicomponent population-based evolutionary algorithms [8]–[11] use numerous points to search for the contour set Λ. As the algorithms progress, new points are generated to replace existing points in such a way as to explore the landscape E. Multicomponent search methods are typically more computationally demanding than single-component methods but offer the advantages of being less sensitive to both local minima and initial conditions.

1) Genetic Algorithms: Eberhart and Dobbins [5] were the first to suggest an evolutionary approach for the inversion of a trained feedforward perceptron with hidden nodes. They applied a standard genetic algorithm [11] for neural network inversion. The basic operations of the genetic algorithm on the search points are: 1) selection based on fitness; 2) recombination of genetic material based on crossover; and 3) mutation. When applied to neural network inversion, each network input search point was encoded as a single bit string. A fitness function, such as

$$\exp(-\alpha E(\mathbf{i})) \tag{6}$$

where α is a positive constant, returns a fitness score based on the quality of each search element. There is no provision in the basic genetic algorithm, however, to assure the solution points are evenly dispersed on Λ.

2) Boundary Marking—An Evolutionary Algorithm for Inversion: Reed and Marks [6] proposed an evolutionary algorithm for neural network inversion that seeks to evenly distribute points on the manifold described by $f(\mathbf{i}) = c$. The selection process allows only the fittest individuals to have a chance of producing offspring and only a subset of the population is replaced during each generation. A step is added whereby the most crowded points are moved apart to help spread them evenly across the inversion manifold. Convergence is illustrated in Fig. 8. The algorithm, dubbed boundary marking, is governed by two basic goals.

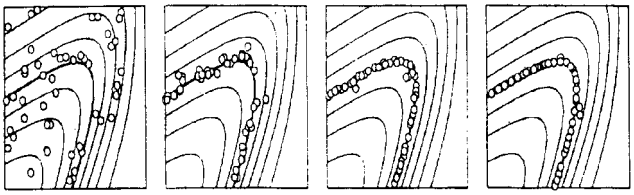


Fig. 8. Repelling crowded search points. The vector $\mathbf{i}_j$ is moved away from its crowding nearest neighbor, $\mathbf{i}_k$, perpendicular to the gradient at $\mathbf{i}_j$.

1) The points should lie on the surface $f(\mathbf{i}) = c$.
2) The points should be evenly distributed over the manifold surface.

The first goal is achieved by periodically eliminating points that are distant from the inversion contour. The second goal is achieved by generating point replacements using perturbations of the least crowded remaining search elements. The second step discourages reproduction in regions that are already crowded with search elements and encourages exploration of sparsely populated regions close to the contour.

After randomly generating N points in the input search space of interest, the algorithm iterates as follows.

1) *Evaluation:* Sort the points by their $E(\mathbf{i})$ errors.
2) *Selection:* Delete the M search points with the largest errors.
3) *Repopulation:* Generate a replacement for each deleted point.
   a) Sort the remaining points in order of d_j^m, their average distance to their nearest m neighbors. A common distance measure is the Euclidean distance but, in general, any distance metric may be used.
   b) Select a parent k from the least-crowded points. Random selection from the first $N/5$ points in the d_j sort order is typical.
   c) Generate the new point in the input space, $\mathbf{i}_{k,\,\mathrm{new}} = \mathbf{i}_{k,\,\mathrm{new}} + \mathbf{n}$ where $\mathbf{n}$ is a random vector [e.g., zero mean independently, identically, distributed (i.i.d.) Gaussian random numbers with variance σ^2]. During the initial stages, σ is chosen to be sufficiently large to explore the search space. In later stages, σ may be decreased to focus the search.
4) *Repelling Crowded Points:*
   a) For each of the most crowded search points (e.g., $N/5$), calculate a vector away from its nearest

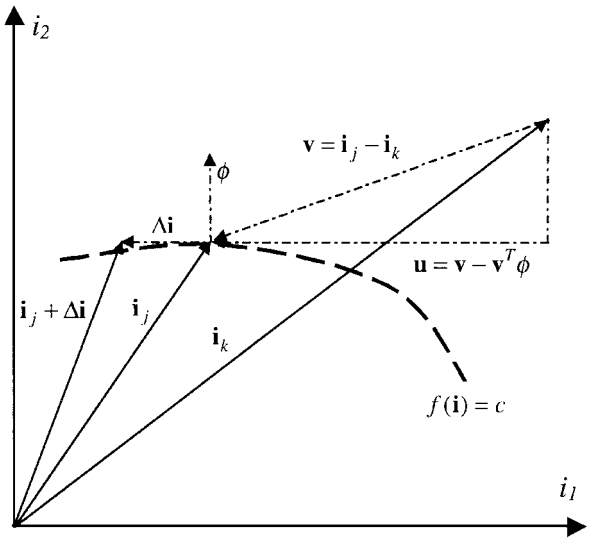


Fig. 9. Illustration of various stages of the evolutionary boundary marking in two dimensions. The search points, denoted by hollow circles, attempt to evenly disperse along the contour. The parameters used in this example are $N = 50$, $M = 3$, $m = 2$, $s = 0.01$, and $\sigma = 0.05$ (from [6]).

neighbor. If $\mathbf{i}_j$ is the point, then the vector is

$$\mathbf{v} = \mathbf{i}_j - \mathbf{i}_k \tag{7}$$

where $\mathbf{i}_k$ is the nearest neighbor of $\mathbf{i}_j$. A unit gradient vector, ϕ, is computed at $\mathbf{i}_j$ using

$$\phi = \frac{\mathbf{g}}{||\mathbf{g}||} \tag{8}$$

where the gradient is

$$g(\mathbf{i}) = \frac{\partial f}{\partial \mathbf{i}}. \tag{9}$$

b) The vector $\mathbf{u}$ is the component of $\mathbf{v}$ perpendicular to $\mathbf{g}$, $\mathbf{u} = \mathbf{v} - \mathbf{v}^T\phi$. The perturbation imposed on $\mathbf{i}_k$ is

$$\Delta\mathbf{i} = s\frac{\mathbf{u}}{||\mathbf{u}||} \tag{10}$$

where s is a step size, i.e., the vector $\mathbf{i}_j$ is moved to $\mathbf{i}_j + \Delta\mathbf{i}$. Since the move is perpendicular to the gradient, we generally expect

$$f(\mathbf{i}_j) \approx f(\mathbf{i}_j + \Delta\mathbf{i}). \tag{11}$$

This process is illustrated in Fig. 9.

5) Go to step 1) and repeat until desired convergence. Various stages of convergence of a two-dimensional example are shown in Fig. 8.

3) Query-Based Learning: Query learning [1] is a process whereby a partially trained neural network is inverted to determine inputs points wherein the classification is uncertain. These points are then verified via an oracle that always responds with the correct answer. An oracle is an expensive source of highly accurate training data, such as a computationally intensive emulator or a data gathering field trip.

If the output layer of a feedforward neural network is passed through a sigmoid nonlinearity, and each output lies between zero and one. A threshold, e.g., $c = 0.5$, is typically applied in binary classification to determine the classification of the input. The most ambiguous points in the classification are those that are close to the classification boundary. The classification boundary is the manifold defined by the locus of input points for which the output is the threshold c. Conversely, those points that are far removed from the boundary tend to be classified more accurately. Therefore, in the absence of pronounced jitter [13], the addition of training points that lie close to the classification boundary results in the greatest potential improvement in classification accuracy. Boundary marking is a technique whereby evenly spaced boundary points can be generated for eventual presentation to the oracle.

Query-based learning has been applied to a number of applications, include cytology screening [16], [17], power system security assessment [18]–[20], and classification of incomplete data [23].

Example 2—Query-Based Learning Applied to Power System Security Assessment: An application of neural network query learning using boundary marking is in the field of power system security assessment. This deals with the ability of a large-scale electric power system to deliver a continuous uninterrupted supply of electric energy to consumers. An interruption in the supply of electric energy can have serious detrimental effects on consumers ranging from the loss of basic living requirements such as heating and lighting for residential customers, to the loss of entire production runs for industrial customers. The goal of power system security assessment is to determine if an electric power system will be able to continue to deliver energy following a contingency, which is defined as a disturbance to the system and can range from a sudden large change in system load to a short circuit on a high-voltage transmission line.

The equation of motion of the rotor of a synchronous generator connected to an electric power system is given by

$$\frac{2H}{\omega_0}\frac{d^2\delta}{dt^2} = T_m - T_e - K_D\frac{d\delta}{dt} \tag{12}$$

which is commonly referred to as the swing equation and represents the movement of the rotor angle δ during a disturbance. The defining variables are:

- H inertia constant;
- ω_0 initial angular velocity of the rotor;
- δ angular position of the rotor with respect to a rotating reference;
- T_e electromagnetic torque;
- T_m mechanical torque;
- K_D damping coefficient.

A contingency causes a sudden change in the electrical characteristics of the system, which in turn causes a disturbance to the electromechanical torque on the shafts of nearby generators. The disturbance causes the system

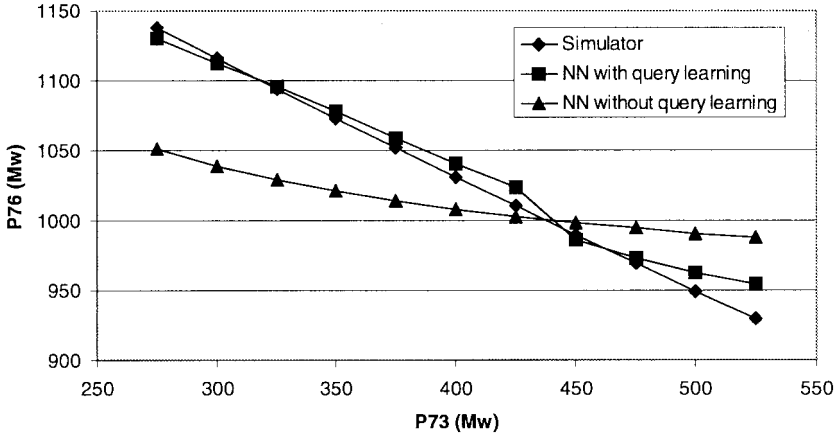


Fig. 10. The power system security boundary predicted by the neural network before and after query learning compared to the actual security boundary determined by the simulation software. Initially, the rms error between the security boundary defined by the neural network and the simulator was 48.53, query learning improved the accuracy of the neural network and reduced this error to 10.11. The axes of this figure represent the power output of the generators on buses 73 and 76, respectively.

to enter a transient period where one or more generators experience oscillations. The oscillations will either increase or decrease depending on the initial conditions and the damping of the system. If the oscillations are not sufficiently damped, one or more machines will be disconnected by their protection systems, resulting in a further disturbance to the system. This is called a cascading outage and may lead to a breach in system security such as a blackout.

Neural networks have shown promise in the fast screening and detection of operating conditions that may lead to blackouts [19], [20]. In these applications, neural networks are trained to classify a power system as secure or insecure based on a set of features of the prefault system such as transmission line flows, generation settings, or system voltage levels. The neural networks are then used during the everyday operation of the system to assess the security status of the system.

The system studied is based on the IEEE 17-generator transient stability test system [22]. It consists of 17 generators and 162 buses. Classical machine data are available for each of the 17 generators. The data consist of the rotor inertia time constant and transient reactance. The first swing stability criterion is used to classify the system stability. The criterion monitors the rotor angle of each generator during the first oscillation. If any generator angle loses synchronism with the rest of the system, the system is said to be unstable. The Extended Transient/Mid-Term Stability Program (ETMSP) from The Electric Power Research Institute (EPRI) is used to conduct the simulations [21]. This software can accurately simulate the dynamic response of very large scale power systems to a number of contingencies.

The neural network used in this study consists of 12 inputs and a single output. The 12 inputs are the real and reactive outputs of the five generators that are closest to the disturbance and the system's real and reactive load level. The neural network was then trained with a small set of training data (100 patterns) followed by application of the query learning algorithm to refine the accuracy of the neural network near the classification boundary. The classification boundary was defined by a neural network output of 0.33 corresponding to a critical clearing time of 0.33 s. Several different network topologies were experimented with and the best results were obtained with a single hidden layer with ten neurons.

Results of the query learning algorithm applied to power system security assessment are shown in Fig. 10. The security boundary as defined by the neural network is plotted in two dimensions and compared with the actual security boundary that was determined from the simulator. The experiment involved training two neural networks, each with 500 patterns. The training data for the first network was created by randomly perturbing the system and then determining the corresponding security level by simulating perturbations. The training data for the second neural network was generated via the evolutionary boundary-marking algorithm explained in Section II-C2. Specifically, an initial network was trained on 100 patterns generated from the simulator. Next, the boundary-marking algorithm was used to evenly spread 400 additional data on the security boundary as defined by the initial network. The simulator was then used to determine the true security margin of these data and the network was retrained. The average multirun rms error for the neural network trained via the query learning algorithm was 10.11 compared with an average of the 48.53 for networks trained without query learning.

III. Enforcement of Constraints in Neural Network Inversion

Many neural network inversion applications require the imposition of constraints on the solution. Therefore, the inversion must not only lie on Λ, but also within a given constraint region. For this case, we want to find the set of

points

$$\Lambda_\Phi = \{\mathbf{i}\colon \mathbf{i} \in \Phi,\ f(\mathbf{i}) = c\} \tag{13}$$

where Φ is the constraint set. Λ_Φ is the intersection of the set of points obeying the inversion and the constraint set

$$\Lambda_\Phi = \Lambda \cap \Phi. \tag{14}$$

During inversion, constraints can be enforced in two ways:

1) internally through modification of the objective or fitness function;
2) externally through iterative interaction with a constraint operation.

Range constraints, for example, can be enforced through the modification of the objective function or by simply clipping the search points during the inversion to the desired constraint region. More complex constraints are typically enforced through iteration with a constraint function. The inversion method and the application usually dictate the choice of the constraint enforcement mechanism.

A. Objective Function Modification for Single-Element Search

The input layer in most networks does not make use of a nonlinear activation function, e.g., a sigmoid function. Therefore, the direct solution of (4) can result in undesirably large components in the solution vector $\mathbf{i}$. Linden and Kinderman [3] proposed a method to enforce limits on the input vector $\mathbf{i}$ during single element inversion such that $\mathbf{i}^{\min} \le \mathbf{i} \le \mathbf{i}^{\max}$. A fictitious input activation function $g(\cdot)$ is inserted in the network input. This function has a limited output range of $(\mathbf{i}^{\min}, \mathbf{i}^{\max})$. A reasonable choice for $g(\cdot)$ is the modified sigmoid function

$$\mathbf{x} = g(\mathbf{i}) = \frac{\mathbf{i}^{\max} - \mathbf{i}^{\min}}{1 + e^{-\mathbf{i}}} - \mathbf{i}^{\min}. \tag{15}$$

Performing gradient descent in the $\mathbf{x}$ space and then transforming the solution back to the $\mathbf{i}$ space via (9) guarantees the inversion will be constrained to the hypercube defined by $(\mathbf{i}^{\min}, \mathbf{i}^{\max})$.

Linden and Kinderman also introduce the concept of attractors to neural network inversion. Attractors tend to pull the inversion toward a desired input pattern. A modified error function can be written as

$$\hat{E} = E + \lambda \sum (a_k - |i_k|)^2 \tag{16}$$

where a_k is the kth attractor and λ is the strength of the attraction. Setting $a_k = 0$ tends to pull the solution vector toward zero resulting in an input decay similar to the standard weight decay [7]. By setting $a_k = 1$, the inputs' activations are pulled toward the binary values of ± 1 which is useful for inverting to binary inputs. Similar modifications can be made to minimize or maximize the input activation level [3], [12]. It should be noted that when the modified error function (16) is used, the optimization is being performed in the $\hat{E}$ space and the solutions may no longer be local minima of E.

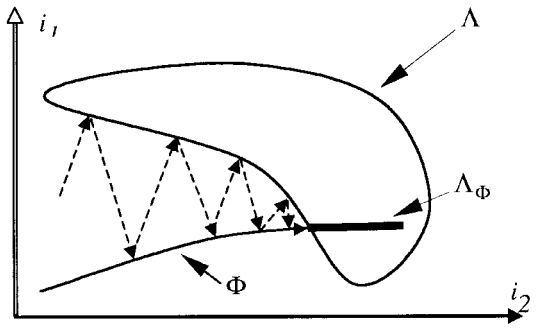


Fig. 11. The solution to the constrained inversion problem can be obtained by iteratively inverting the net to find an element in Λ, followed by a projection onto the constraint set Φ. The solution set Λ_Φ, shown by a thick line, is the intersection of the set of allowable inversion points, Λ, and the constraint set Φ.

B. Objective Function Modification for Multi-Element Search

Evolutionary methods offer much flexibility in the enforcement of constraints. Typically a penalty term is added to the fitness function to penalize solutions that violate the constraints. The penalty term allows the algorithm to explore regions of the input space that are in violation of the constraint in the hope of discovering new regions that provide better solutions. The penalty term also allows the solution to converge to a compromise between constraint satisfaction and inversion solution in the event of an empty Λ_Φ.

C. Enforcement of Constraints via Alternating Projections

Constraints can also be imposed on the inversion by the interaction of the inversion algorithm with a constraint enforcement procedure. Hwang *et al.* [14] proposed an iterative method similar to the alternating projection onto convex sets algorithm [15] to find a point that is a member of the intersection of Λ and the input constraint set Φ. The alternating projection solution is found by iteratively inverting the network to Λ and then projecting onto the constraint set Φ. The process is shown for a two-input, one-output network in Fig. 11.

The algorithm proceeds as follows.

1) Starting with an initial input, use a nearest inversion single-element search algorithm to find a point on Λ.
2) This point, in general, will not be in the constraint set Φ. The point is therefore projected onto the constraint set.[2]
3) Using the point on the constraint set as initialization, nearest inversion results in a new point in the set Λ.
4) Return to step 2) and repeat until the desired convergence is attained.

Convergence of the alternating projections procedure to a point in the intersection is guaranteed if both Λ and Φ are convex or, more generally, if the cascading of the operations in steps 2) and 3) form a composite contractive

[2] A projection onto a set of points is defined as the point that is closest to the original point in the mean square sense.

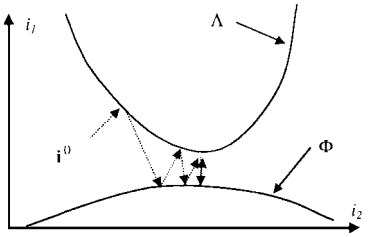


Fig. 12. When the inverse set Λ and the constraint set Φ are both convex, alternating projections will reach a limit cycle between two points—each a point in one set that is closest to the other.

operator [15]. Even though convergence cannot generally be guaranteed, remarkably good results occur often. An example is given in Section IV.

The above approaches assume that the set Λ_Φ is not empty. In general, if Λ_Φ is empty and Λ and Φ are convex [15], the alternating projections will converge to a point that does not satisfy the desired objective. The constraint can then either be relaxed or the suboptimal solution can be accepted. If the iteration continues to convergence and is stopped on constraint set Φ, the result is a point in Φ that lies closest to the inversion set, Λ [15]. This is illustrated in Fig. 12.

Example 3: Jensen *et al.* [18] proposed the use of alternating projections using neural network inversion as a means to identify and track the security boundary for large-scale electric power systems. More specifically, a neural network is trained to assess the security of the large-scale electric power system described in Example 2 based on a set of 33 prefault features. The features include generator real and reactive power outputs as well as the total system load. The security index is based on the concept of critical clearing time (a measure of the maximum allowable fault duration for a specific fault). The longer a fault can exist on the system before instability occurs, the higher the critical clearing time and, hence, the more stable the system is said to be.

Constrained neural network inversion is used to locate the security boundary relative to the current operating state of the system. Knowledge of the exact location of the security boundary relative to a given operating configuration is useful in the everyday operation of the system because it provides operators with the ability to steer the system away from insecure regions. In the event of a breach of system security, knowledge of the exact location of the boundary can also be used to regain security.

The neural network inversion was performed using the nearest inversion procedure [25] discussed in Section II-A1. As shown in Fig. 13, the algorithm begins by projecting a randomly generated search point onto the security boundary and then performs a constrained gradient descent on the security boundary to locate the point that is closest to the current operating state. Repeated searches were performed starting from different perturbations of the current operating point to increase the likelihood of finding the global minima and to identify the security boundary in several directions. Electric power flow constraints were enforced by iterating

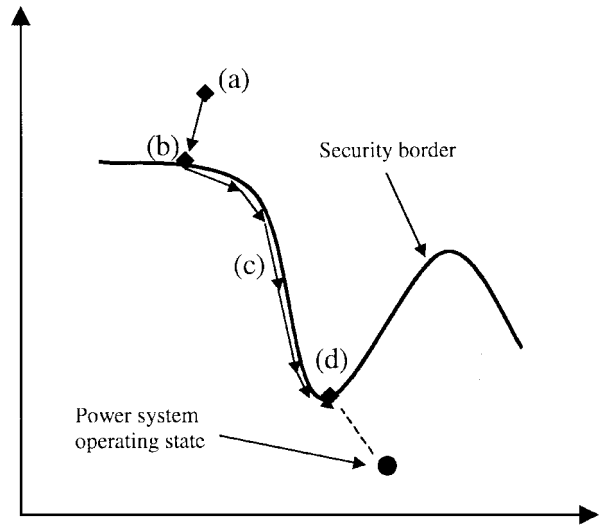


Fig. 13. The power system security border is identified by performing repeated searches starting from (a) perturbations of the current operating state. Each search consists of (b) a gradient-based neural network inversion to locate the security border, followed by (c) a constrained gradient descent on the security border to locate the operating point closest to (d) the current power system operating point.

with an external power flow simulation program. This ensures that the inversion converges to a feasible power system operating point that obeys Kirchhoff's laws.

Jensen *et al.* [18] conducted an experiment where the constrained inversion algorithm is performed on 100 randomly generated power system operating states. The solution points verified by the simulation software were found to be feasible power system operating states. In an attempt to verify that the inversion found the nearest boundary point, a local search for a superior[3] solution was conducted by repeatedly adding random noise to the solution points and simulating the resulting points. In no case was a superior solution found, thus verifying that the solutions found from the constrained inversion are locally optimal.

IV. Other Applications of Neural Network Inversion

A. Extraction of Codebook Vectors

For a trained neural network, a codebook vector is "an input pattern that results in a maximum or nearly maximum activation value for a given output neurode" [5]. The extraction of codebook vectors is useful for revealing characteristics of the input space that are important to a particular output vector, i.e., act as an explanation facility to aid the user in better understanding why a network reaches a particular decision.

Eberhart and Dobbins [5] describe an application where a neural network is trained to perform the diagnosis of appendicitis. The researchers were interested in which combinations of input features made up the quintessential

[3] A superior solution corresponds to a solution that lies closer to the given search point and satisfies the given constraints.

cases of appendicitis and nonspecific abdominal pain as well as what cases represent the decision surface.

The neural network used to address the diagnosis problem consisted of 106 binary input features, some of which represented the presence or absence of particular symptoms. Other features were used in combinations to represent fields of information such as age. The network was trained to discriminate between appendicitis and nonspecific abdominal pain. Cases of appendicitis were assigned a binary one and nonspecific abdominal pain were assigned a zero. The threshold for appendicitis was determined to be 0.78.

The neural network was then inverted using a genetic-based inversion algorithm. Several codebook vectors for each of the above cases were extracted from the trained neural network by inversion with the output set to appendicitis (1.0), nonspecific abdominal pain (0.0), and the decision surface (0.78), respectively. A physician then reviewed the codebook vectors and found them to be "reasonable and consistent," thereby increasing confidence in the networks' ability to accurately diagnose appendicitis.

In another application, Linden and Kindermann [3] trained a neural network to classify the handwritten digits 0–9. The digits were coded as 11×8 bitmaps. The neural network consisted of $11 \times 8 = 88$ inputs, 20 hidden units, and ten output units representing the digits 0–9. The network was trained with 49 sets of the ten digits for a total of 490 training patterns. Codebook vectors for each digit were then extracted by inversion using gradient descent with the output corresponding to the particular digit of interest. The codebook vectors were then compared to typical examples of each digit to assess the networks' generalization.

B. Model Reference Adaptive Control

Hoskins *et al.* [24] proposed the use of neural network inversion as a means to find a control input that causes a plant to respond with desired error perturbation dynamics. The system consists of several components (see Fig. 14), the first being a performance model which is a discrete time representation of the desired closed loop system behavior in response to the command input, $r(t_k)$. The second is the convergence model that determines the desired perturbation dynamics of the true plant about the output of the performance model. The last component is a controller based on a neural network trained to the nominal forward dynamics of the system. The neural network model is updated online via backpropagation learning to adapt to changing plant dynamics.

The action of the controller follows. The performance model generates a reference signal and the convergence model specifies the desired perturbation behavior of the system. A Lyapunov function is found based on the convergence model. The neural network is then inverted to find the control input that minimizes the difference between the predicted and desired values of the Lyapunov function. This technique has the advantage over existing neural network-based control methods in that it removes the highly nonlinear neural network from the direct feedback path, thus simplifying analysis [24].

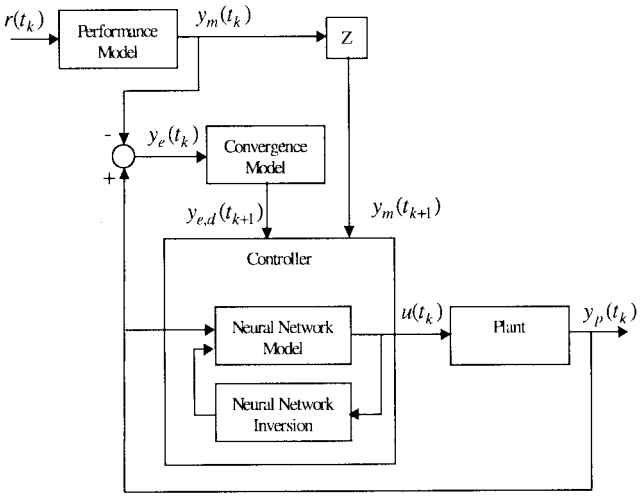


Fig. 14. Neural network model reference adaptive controller (NN-MRAC) structure.

C. Additional Applications

1) Martin and Millan [26] and Behera *et al.* [27] use neural network inversion as an aid in the control of multilink robot manipulators. A feedforward neural network is trained to approximate the forward kinematics of the manipulator arm. Martin and Millan [26] invert the trained neural network to produce a goal vector which specifies the direction the robot arm should follow to approach a desired position. Behera *et al.* [27] propose a similar control strategy using an extended Kalman filter-based neural network inversion scheme.
2) A frequency selective surface (FSS), consisting of a periodically arranged identical metallic patched or aperture elements supported by dielectric layers on, for example, an aircraft's surface, exhibits varying electromagnetic wave reflection properties for different incident frequencies. The forward problem of computing the frequency response of a given FSS is well understood. The inverse design problem is to find an FSS for a desired frequency response. The traditional method of FSS design is a trial-and-error manual search of a knowledge base until a surface is found with the desired characteristics. The process is laborious and tedious. Hwang *et al.* [14] use constrained neural network inversion as a method for automated FSS design. The process begins with a neural network that has been trained to learn the one-dimensional (1-D) frequency responses of various two-dimensional (2-D) FSS's. Given a desired frequency response, a constrained inversion scheme is used to invert the neural network to produce a FSS with desired characteristics.
3) Neural network inversion is proposed as a means of controlling the growth of crystals by Ishida and

Zhan [30]. The crystalline growth process requires a specific temperature profile throughout the growth. The temperature is controlled via several electric heating elements positioned in the growth chamber. It is desired to predict the required control signal for each of the heating elements so that the temperature profile throughout the chamber follows a desired track. This is accomplished by first learning the forward problem of predicting the temperature, T, at time $t+1$ given the temperature and the heat supply rate q at time t for each of the heaters. The trained neural network is then inverted with the outputs fixed to the desired temperature at time $t+1$ to determine the required heat rate q at time t.

4) Weijer *et al.* [32] trained a neural network to learn the relationship between the physical structure of polyethylene terephthalate (PET) yarns and their mechanical and shrinkage properties. The trained neural network is inverted to find the physical structure(s) that correspond to a desired set of shrinkage properties. The inversion procedure uses a genetic algorithm approach.

V. Conclusions

Trained feedforward perceptron neural networks are often able to capture both the forward and inverse mappings of the underlying process that generated the training data. The trained neural network can be inverted using a number of optimization techniques including single-element and multi-element (population-based) evolutionary approaches. Evolutionary techniques, in particular, are useful when a number of inversion points are desired. Neural network inversion has proved to be a valuable in tool in numerous engineering applications.

The best choice of inversion algorithm to use is problem specific. Single-element inversion is appropriate in problems where any single inverse solution suffices. For instance, in the sonar example, any inversion solution that gives sufficient signal-to-interference ratio is an acceptable answer. For the problem presented, there is no need to generate and compare a number of inverse points, therefore, use of single element inversion is appropriate. A disadvantage of single-element methods, besides the possibility of premature convergence due to local minima, is the increased difficulty in imposing constraints on the solution. Although there are specific cases where iteration between a steepest descent solution and constraint projections are assured to be stable, the cases are both difficult to prove and, in our experience, unlikely to occur.

Premature convergence becomes probable when multiple local minima exist. In such cases, the inversion algorithm must be restarted from numerous different starting points to assure the global minimum is detected. Multi-element inversion is appropriate for problems that: 1) require multiple solutions; 2) involve numerous local minima; or 3) require the enforcement of complex constraints. The principal disadvantage of multi-element inversion methods is, due to multiple solutions being required each generation, the comparatively slow solution speed. Inversion to a plurality of solutions is also possible through repetition of sequential single element inversions using different initializations. Our experience is, however, that single-element inversion often results in several unique basins of attraction where large regions of initialization result in similar inverse solutions, while other inverse solutions remain illusive from any initialization.

While this paper has concentrated on inverting feedforward perceptron neural networks, the inversion methods described herein can also be applied directly to memoryless time-invariant (i.e., static) nonlinear functions with multiple inputs and outputs. For nonlinear systems with memory, the mapping at any point in time is a function of prior inputs. Inversion of recurrent neural networks and similar nonlinear systems with memory is a more ominous problem—one not yet addressed in the literature. Similarly, dynamic (i.e., time-variant) nonlinear systems—such as neural networks with temporally adjusting weights—have mappings that change with time. Their inversion is likewise more problematic.

References

[1] J. N. Hwang, J. J. Choi, S. Oh, and R. J. Marks, II, "Query based learning applied to partially trained multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 2, pp. 131–136, 1991.

[2] R. J. Williams, "Inverting a connectionist network mapping by backpropagation of error," in *Proc. 8th Annu. Conf. Cognitive Science Society.* Hillsdale, NJ: Lawrence Erlbaum, 1986, pp. 859–865.

[3] A. Linden and J. Kindermann, "Inversion of multilayer nets," in *Proc. Int. Joint Conf. Neural Networks*, vol. II, Washington, DC, 1989, pp. 425–430.

[4] J. N. Hwang, J. J. Choi, S. Oh, and R. J. Marks II, "Classification boundaries and gradients of trained multilayer perceptrons," in *Proc. IEEE Int. Symp. Circuits and Systems*, 1990, pp. 3256–3259.

[5] R. C. Eberhart and R. W. Dobbins, "Designing neural network explanation facilities using genetic algorithms," in *Proc. Int. Joint Conf. Neural Networks*, vol. II, Singapore, 1991, pp. 1758–1763.

[6] R. D. Reed and R. J. Marks, II, "An evolutionary algorithm for function inversion and boundary marking," in *Proc. IEEE Int. Conf. Evolutionary Computation (ICEC'95)*, Perth, Western Australia, 1995, pp. 794–797.

[7] ——, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks.* Cambridge, MA: MIT Press, 1999.

[8] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* Piscataway, NJ: IEEE Press, 1995.

[9] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. Berlin, Germany: Springer-Verlag, 1996.

[10] D. B. Fogel, Ed., *Evolutionary Computation: The Fossil Record.* Piscataway, NJ: IEEE Press, 1998.

[11] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison-Wesley, 1989.

[12] J. Linderman and A. Linden, "Inversion of neural networks by gradient descent," *Parallel Comput.*, pp. 277–286, 1990.

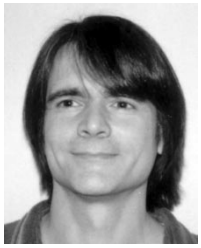
[13] R. Reed, R. J. Marks II, and S. Oh, "Similarities of error regularization, sigmoid gain scaling, target smoothing and training with jitter," *IEEE Trans. Neural Networks*, vol. 6, pp. 529–538, May 1995.

[14] J. N. Hwang, C. H. Chan, and R. J. Marks II, "Frequency selective surface design based on iterative inversion of neural networks," in *Proc. Int. Joint Conf. Neural Networks (IJCNN '90)*, vol. 1, San Diego, CA, 1990, pp. 39–44.

[15] R. J. Marks II, "Alternating projections onto convex sets," in *Deconvolution of Images and Spectra*, P. A. Jansson, Ed. San Diego, CA: Academic, 1997, pp. 476–501.
[16] D. T. Davis, J. N. Hwang, and J. S. Lee, "Improved network inversion technique for query learning: Application to automated cytology screening," in *Proc. Computer-Based Medical Systems: 4th Annu. IEEE Symp.*, Baltimore, MD, 1991, pp. 313–320.
[17] D. T. Davis and J. N. Hwang, "Attentional focus training by boundary region data selection," in *Proc. Int. Joint Conf. Neural Networks, (IJCNN'92)*, Baltimore, MD, June 1992, pp. 676–681.
[18] C. A. Jensen, R. D. Reed, M. A. El-Sharkawi, and R. J. Marks II, "Location of operating points on the dynamic security border using constrained neural network inversion," in *Proc. Int. Conf. Intelligent Systems Applications to Power Systems, (ISAP'97)*, Seoul, Korea, July 1997.
[19] M. A. El-Sharkawi and S. S. Huang, "Query-based learning neural network approach to power system dynamic security assessment," in *Proc. 1993 Int. Symp. Nonlinear Theory and Its Applications (NOLTA'93)*, Hawaii, Dec. 1993.
[20] ——, "Application of query-based learning to power system static security assessment," in *Proc. 2nd Int. Forum on Applications of Neural Networks to Power Systems (ANNPS'93)*, Yokohama, Japan, Apr. 1993, pp. 111–117.
[21] EPRI, "Extended transient midterm stability program: Version 3.0," Palo Alto, CA, EPRI TR-102004, vol. 1-6, 1993.
[22] IEEE Committee Report, "Transient stability test systems for direct stability methods," *IEEE Trans. Power Syst.*, vol. 7, pp. 37–44, Feb. 1992.
[23] J. N. Hwang and C. J. Wang, "Classification of incomplete data with missing elements," in *1994 Int. Symp. Artificial Neural Networks, (ISANN'94)*, Tainan, Taiwan, Dec. 1994, pp. 471–477.
[24] D. A. Hoskins, J. N. Hwang, and J. Vagners, "Iterative inversion of neural networks and its application to adaptive control," *IEEE Trans. Neural Networks*, vol. 3, pp. 292–301, Mar. 1992.
[25] R. D. Reed, R. J. Marks, II, C. A. Jensen, and M. A. El-Sharkawi, "A neural network inversion procedure," in *Int. Joint Conf. Neural Networks (IJCNN'98)*, Anchorage, AK, 1998.
[26] P. Martin and J. R. Millan, "Combining reinforcement learning and differential inverse kinematics for collision-free motion of multilink manipulators," in *Int. Work-Conf. Artificial and Natural Neural Networks, (IWANN'97)*, Lanzarote, Spain, June 1997, pp. 1324–1333.
[27] L. Behera, M. Gopal, and S. Chaudhury, "On adaptive trajectory tracking of a robot manipulator using inversion of its neural emulator," *IEEE Trans. Neural Networks*, vol. 7, no. 6, Nov. 1996.
[28] K. Yoshitomi, A. Ishimaru, J. N. Hwang, and J. S. Chen, "Surface roughness determination using spectral correlations of scattered intensities and an artificial neural network technique," *IEEE Trans. Antennas Propagat.*, vol. 41, pp. 498–502, Apr. 1993.
[29] M. Sase, N. Kinoshita, and Y. Kosugi, "A neural network for fusing the MR information into PET images to improve spatial resolution," in *Proc. (ICIP'94)*, vol. 3, Austin, TX, Nov. 1994, pp. 908–911.
[30] M. Ishida and J. Zhan, "Neural model-predictive control of distributed parameter crystal growth process," *AICHE J.*, vol. 41, no. 10, pp. 2333–2336, Nov. 1995.
[31] G. C. Vasconcelos, M. C. Fairhurst, and D. L. Bisset, "Enhanced reliability of multilayer perceptron networks through controlled pattern rejection," *Electron. Lett.*, vol. 29, no. 3, pp. 261–263, 1993.
[32] A. P. Weijer, C. B. Lucasius, L. Buydens, and G. Kateman, "Using genetic algorithms for an artificial neural network model inversion," *Chemometrics Intell. Lab. Syst.*, vol. 20, no. 1, pp. 45–55, Aug. 1993.
[33] J. N. Hwang and C. H. Chan, "Iterative constrained inversion of neural networks and its applications," in *Proc. 24th Conf. Inform. Sci. Syst.*, Princeton, NJ, Mar. 1990, pp. 754–759.
[34] J. Takeuchi and Y. Kosugi, "Neural network representation of finite element method," *Neural Networks*, vol. 7, no. 2, pp. 389–395, 1994.
[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing (PDP): Exploration in the Microstructure of Cognition.* Cambridge, MA: MIT Press, 1986, vol. 1.
[36] M. I. Jordan and D. E. Rumelhart, "Forward models: Supervised learning with a distal teacher," *Cognitive Sci.*, vol. 16, pp. 307–354, 1992.
[37] M. I. Jordan, "Constrained supervised learning," *J. Math. Psych.*, vol. 36, pp. 396–425, 1992.
[38] B. L. Lu, H. Kita, and Y. Nishikawa, "Inversion of feedforward neural networks by a separable programming," in *Proc. World Congr. Neural Networks (Portland)*, vol. 4, 1993, pp. 415–420.
[39] ——, "A new method for inverting nonlinear multilayer feedforward networks," in *Proc. Int. Conf. Industrial Electronics, Control and Instrumentation*, Kobe, Japan, 1991, pp. 1349–1354.

Craig A. Jensen received the B.S.E.E. and M.S.E.E. degrees from the University of North Dakota, Grand Forks, in 1993 and 1995, respectively, and the Ph.D. degree from the University of Washington, Seattle, in 1999.

He was employed at Otter Tail Power Company in Fergus Falls, MN, as an Engineering Intern in the System Planning and System Engineering Departments. His research interests are neural networks, computational intelligence, intelligent systems applications to power systems, and computer programming.



Russell D. Reed received the B.S. and M.S. degrees in electrical engineering from Texas A&M University, College Station, TX, in 1981 and 1986, respectively, and the Ph.D. degree in electrical engineering from the University of Washington, Seattle, in June 1995.

In the past, he has served as a Research Assistant Professor in the Department of Electrical Engineering, University of Washington. He is currently with Bass Software, Longview, TX. He is a coauthor of *Neural Smithing, Supervised Learning in Feedforward Artificial Neural Networks* (MIT Press, 1999). His research interests include neural networks, pattern recognition, machine learning, biologically inspired computational mechanisms, and artificial intelligence.

Robert J. Marks, II (Fellow, IEEE) is a Professor and Graduate Program Coordinator in the Department of Electrical Engineering, College of Engineering, University of Washington, Seattle. He is the author of numerous papers and is coauthor of the book *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks* (MIT Press, 1999). He currently serves as the faculty advisor to the University of Washington's chapter of Campus Crusade for Christ. His hobbies include cartooning, song writing, *Gunsmoke*, and creating things for his website.

Dr. Marks is a Fellow of the Optical Society of America. He served as the first President of the IEEE Neural Networks Council. In 1992 he was given the honorary title of Charter President. He served as the Editor-in-Chief of the IEEE TRANSACTIONS ON NEURAL NETWORKS and as a Topical Editor for Optical Signal Processing and Image Science for the *Journal of the Optical Society on America A*. For more information see: cialab.ee.washington.edu/Marks.html.

Mohamed A. El-Sharkawi (Fellow, IEEE) received the B.Sc. degree in electrical engineering in 1971 from the Cairo High Institute of Technology, Cairo, Egypt, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of British Columbia, Vancouver, B.C., Canada, in 1977 and 1980, respectively.

In 1980, he joined the University of Washington, Seattle, where he is presently a Professor of Electrical Engineering and the Associate Chair. He has also served as the Chairman of Graduate Studies and Research. His major areas of research include intelligent systems applications, high-performance precision drives, and power electronics applications to power systems. He has published more than 120 papers and book chapters in these areas and holds five licensed patents: three on adaptive var controller for distribution systems and two on adaptive sequential controller for circuit breakers. He is the coeditor of the IEEE tutorial book on the applications of neural networks to power systems and author of a forthcoming textbook on electric drives. He has organized and taught several international tutorials on intelligent systems applications, power quality, and power systems.

Dr. El-Sharkawi is Founder of the International Conference on the Application of Neural Networks to Power Systems (ANNPS), cofounder of the International Conference on Intelligent Systems Applications to Power (ISAP), a member of the administrative committee of the IEEE Neural Networks Council representing the Power Engineering Society, Video Tutorial Chair of the IEEE Continuing Education Committee and the IEEE Neural Networks Council, Founding Chairman of several IEEE task forces and working groups and subcommittees (including the task force on Application of Neural Networks to Power Systems, the working group on Advanced Control Strategies for dc-type Machines, and the task force on Intelligent Systems Application to Dynamic Security Assessment), and cofounder of the IEEE Subcommittee on Intelligent Systems. He is a current or past member of the editorial board or Associate Editor of several journals, including IEEE TRANSACTIONS ON NEURAL NETWORKS, *Engineering Intelligent Systems*, and the *International Journal of Neurocomputing*. He was Chairman of the IEEE International Electric Machines and Drives held in Seattle, WA, in May 1999, and he has organized and chaired numerous panel and special sessions in IEEE and other international conferences.

Jae-Byung Jung received the B.S. and M.S. degrees in electronics engineering from Hanyang University, Seoul, Korea, in 1993 and 1995, respectively. He is currently working toward the Ph.D. degree at the University of Washington, Seattle.

His research interests include neural networks, genetic algorithms, fuzzy systems, and other topics related to computational intelligence.

Robert T. Miyamoto received the B.A. degree in mathematics and physics in 1973 from the University of California, Irvine, and the Ph.C. degree (Candidate for Ph.D.) in oceanography in 1976 from the University of Washington, Seattle.

He joined the Applied Physics Laboratory of the University of Washington in 1979. He currently heads the Environmental and Information Systems Department at the same university. His primary area of expertise is interpretation, analysis, simulation, and prediction of acoustic reverberation in the ocean. He is also an expert in estuarine dynamics. He has a thorough knowledge of acoustic models and simulations and a broad understanding of computer operating systems, data-analysis software, and programming languages. His projects involve such problems as: environmental effects on the performance of sensors aboard Navy aircraft, surface ships, and submarines; littoral zone environmental effects on tactical decisions; development and extension of computer models of midfrequency active sonars, acoustic and nonacoustic prediction models, and databases; and development of an algorithm to identify environmental parameters. His other projects include development of an algorithm to identify fish schools through acoustic backscatter images and development of an interactive CD-ROM about the effects of factors such as increased population, pollution, and overfishing on the Puget Sound environment.

Gregory M. Anderson received the B.S. degree in agriculture, the B.S. degree in applied mathematics, and the M.S.E.E. degree from the University of Idaho, Moscow, in 1974, 1975, and 1980, respectively.

In 1990, he joined the Applied Physics Laboratory of the University of Washington, Seattle, where he is currently a Senior Electrical Engineer. He has experience in systems engineering, digital simulation, statistics, optimization, expert systems, and object-oriented programming. He is applying these skills to develop technical information and analysis systems that are easy to use. His current work includes estimating parameters for ocean environmental models from acoustic reverberation measurements, constructing a data storage and analysis system for environmental parameters, and developing a CD-ROM-based system for training firefighters in the handling of hazardous materials.

Christian J. Eggen received the B.S. and M.S. degrees in physics and the M.S.E.E. degree from the University of Washington, Seattle, in 1968, 1975, and 1998, respectively.

Since 1977, he has been with the Applied Physics Laboratory of the University of Washington, Seattle, where he is currently a Principal Physicist. He specializes in sonar technology and simulations. He is an expert in the detection and classification of high-frequency echoes and has worked on the Mk 46 and Mk 50 torpedo projects. He has worked on the application of artificial intelligence, neural nets, and morphological processing to classification problems and on surface modeling. His recent degree focused on signal processing and communications. He has served on numerous Navy technical advisory teams, including the Target Strength Standardization Committee and the Mobile Offboard Sonar Technology Committee. He is currently working on tomography and low-frequency mine-hunting projects.