Inversion of Neural Network Underwater Acoustic Model for Estimation of Bottom Parameters Using Modified Particle Swarm Optimizers

Benjamin B. Thompson, Robert J. Marks II, Mohamed A. El-Sharkawi,
Warren J. Fox, and Robert T. Miyamoto

Computational Intelligence Applications (CIA) Laboratory, Department of Electrical Engineering, University of Washington, Seattle, WA 98195

Applied Physics Laboratory (APL), University of Washington, Seattle ,WA 98105

Abstract

Given a complicated and computationally intensive underwater acoustic model in which some acoustic measurement is a function of sonar system and environmental parameters, it is computationally beneficial to train a neural network to emulate the properties of that model. Given this neural network model, we now have a convenient means of performing geoacoustic inversion without the computational intensity required when attempting to do so with the actual model. This paper proposes an efficient and reliable method of performing the inversion of a neural network underwater acoustic model to obtain parameters pertaining to the characteristics of the ocean floor, using two different modified versions of particle swarm optimization (PSO): two-step (gradient-approximation) PSO and hierarchical cluster-based PSO.

I. Introduction

The forward problem of modeling the acoustical properties of an underwater environment given a set of sonar system parameters and an additional set of environmental parameters can be solved using a typically computationally intensive computer simulation [1]. It is then desirable to invert the model such that, given a particular acoustical state of the underwater environment (e.g., a reverberation measurement), along with the system parameters and certain known environmental parameters, one can obtain information about the unknown environmental parameters [2]-[4]. Due to the computational complexity of the acoustic model, however, this becomes a daunting task to perform in real time.

The application of a feed-forward neural network to emulate the underwater acoustic model reduces the amount of computing power necessary to perform the forward problem, and indeed greatly aids in the task of inverting the model [5]. The problem remains, however, of actually inverting the neural network itself. As will be discussed in greater detail in Section II, a search of the input space is used to determine the "best" set of environmental parameters, where "best" is measured by how well the result matches the measured acoustic properties of the undersea environment. This paper proposes two different methods of inverting the neural network. Both involve the use the particle swarm optimization (PSO) method; however, we suggest two different modifications of the basic PSO algorithm that are shown to provide significant improvement to the performance of the algorithm for this problem [6].

II. Neural Network Inversion

When we refer to "inversion" of a neural network for the acquisition of certain input parameters, we are actually referring to a constrained inversion. That is to say, given the functional relationship of the neural network inputs to the outputs $\vec{y} = f(\vec{x})$, we are not merely trying to find $\vec{x} = f^{-1}(\vec{y})$. Rather, we have the forward and inverse relationships:

$$\vec{y} = f(\vec{x}, \vec{u}) \tag{1}$$

$$\vec{x} = g(\vec{y}_t, \vec{u}), \tag{2}$$

where $\vec{x}$ is the set of unknown environmental parameters we wish to obtain, and $\vec{u}$ is a vector of the

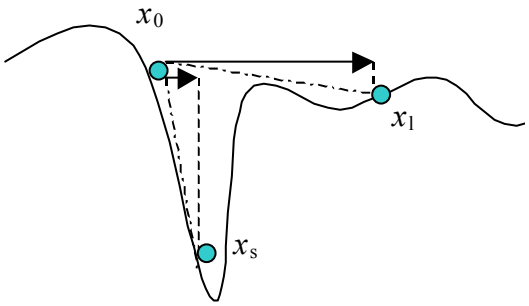

Figure 1 - Global minima which occur in steep, narrow valleys tend to be missed by standard PSO methods. As depicted, a shorter step size would alleviate this problem.

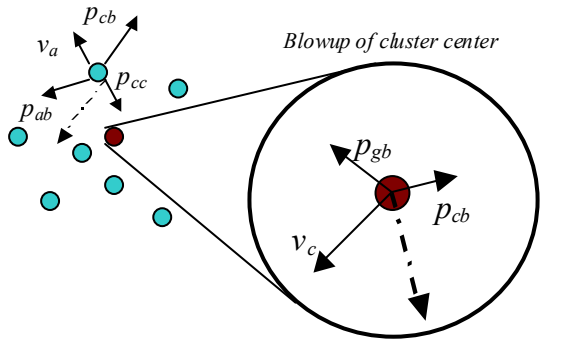

Figure 2 - Basic idea behind CPSO. The dotted line represents the updated trajectory based on each other vector and the weighting

known environmental and system parameters. Given the forward relationship described in a neural network form by (1), we wish to find an inverse mapping described by (2) where $\vec{y}_t$ is a target output measurement obtained through some other method (i.e., it cannot be an output of our model described in (1) since $\vec{x}$ is unknown). The proposed method by which we obtain $\vec{x}$ is via the following relationship:

$$\vec{x}_{est} = \arg\min_{\vec{x}_i}(\| \vec{y}_t - f(\vec{x}_i, \vec{u}) \|^2), \tag{3}$$

where $\vec{x}_{est}$ is the argument that minimizes the error between the target output $\vec{y}_t$ and the neural network output produced by that argument. This scheme essentially sets up the problem as one of optimization; that is, our task is now to find the $\vec{x}$ that optimizes the performance of the neural network in emulating the effects dictated by the target output vector $\vec{y}_t$. For this specific application, we find that particle swarm optimization is very well suited for the problem at hand.

III. Particle Swarm Optimization

Eberhart and Kennedy [6] initially proposed the idea of using multiple agents that behave in a collective, swarm-like manner as a solution to various optimization problems. The basic idea behind PSO is described by the following two equations [7]:

$$v_{id} = w * v_{id} + c_1 * rand(\,) * (p_{id} - x_{id}) + c_2 * rand(\,) * (p_{gd} - x_{id}) \tag{4}$$

$$x_{id} = x_{id} + v_{id}\,. \tag{5}$$

Here, x_{id} is the position of an agent in the search space, v_{id} is its velocity (i.e., the distance it will move from iteration to iteration), p_{id} represents the best position that individual agent has found to date, p_{gd} is the group-best position (among all the agents), and w, c_1, and c_2 are parameters of the system that may be adjusted to improve performance. Also, *rand*() signifies a random number picked from a uniform distribution on [0,1]. It is this element that causes PSO to be a stochastic process, and thus provides us with motivation to perform Monte-Carlo-like simulations in section IV.

Clearly, then, this is an evaluation-only optimization method; that is, no information about the search-space itself is taken into account that might improve the end result. Figure 1 shows a simple situation, however, in which basic PSO methods can fail to converge. As depicted, when the optimal solution lies at the bottom of a narrow valley, it is very easy for each swarm agent to completely "fly" over it without registering an improvement in the global best. We will show later that this situation occurs often in the search spaces in which we are interested. If we were able to take into account some local information about our search space, the optimization algorithm would "know" that, at the initial point x_0, the gradient was very steep, and thus it should proceed in very short steps to find the bottom of the valley (much like a gradient descent technique). Thus, we have motivation for two different modifications over standard PSO that perform significantly better in light of this situation.

A. Two-Step PSO

The first method proposes a two-step approach that essentially approximates the gradient at a local point and decides which of two step sizes to make based on that information (we will refer to this method as PSO2 beyond this point). As anticipated in Figure 1, we see that, in this algorithm, each agent

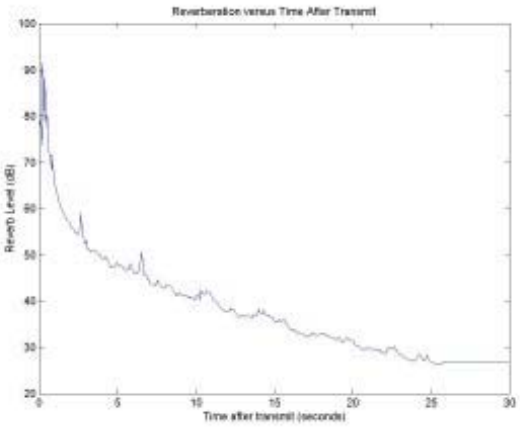

Figure 3 - Typical reverberation measurement in dB as a function of time after transmit, in seconds.

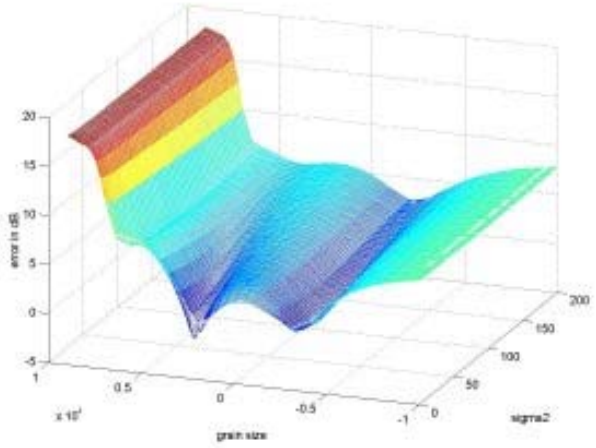

Figure 4 - Typical error surface for two parameters M_z and $\sigma_{2,}$, in dB. Note shallow local minimum and steep valley in which lies the global minimum

takes two steps in the same direction: one in standard PSO fashion, and the other a much shorter distance. Then, the algorithm simply calculates the slope from the initial point x_0 to each point x_s and x_l, and simply chooses the step corresponding to the greatest negative slope. The only additional parameter this introduces into the scheme is some fractional coefficient δ_s, which is the step size relative to the "usual" PSO step size (which could be seen as having its own δ_l=1). This method is summarized in the following additional equations, which assume the initial execution of equation (4):

$$x_l = v_{id} + x_{id} \tag{6a}$$

$$x_s = x_{id} + \delta_s * v_{id} \tag{6b}$$

$$x_{id} = \arg\max_{x_s, x_l}\left(-\frac{F(x_s)-F(x_{id})}{\delta_s}, -\frac{F(x_l)-F(x_{id})}{\delta_l}\right) \tag{7}$$

where $F()$ is the fitness function used to evaluate the position of each agent, assuming the problem is formulated as a minimization scheme.

While this method will be shown to work very well for the problem to be solved in this paper, it is immediately clear that it could be prone to causing premature stoppage of the algorithm due to local minima. The following alternate proposal combines the benefits outlined above of a local search with the local-minima avoidance of a more global technique.

B. Cluster PSO

In an effort to combine both the desired local-search properties to prevent overlooking of the true global minimum with the global-search ability to avoid local minima, we propose the following "cluster"-based modification to PSO. The central idea, as illustrated in Figure 2, is to force a sort of hierarchical structure onto the swarm model, creating clusters of agents that move about the search space while remaining relatively close together. This is in contrast to standard PSO, in which each agent is essentially autonomous. Thus, conceptually, we have the global searching accomplished by each cluster (which, we will show, is designed to behave more or less like a standard particle swarm agent), with the local search performed by the agents within each cluster.

We then have the following expanded set of equations that defines the behavior of this cluster-based PSO extension (CPSO), where (8) and (10) apply to a cluster of agents, and (9) and (11) correspond to the update equations for each individual agent:

$$v_c = w_c * v_c + a_{c_1} * rand(\,) * (p_{cb} - p_{cc}) + a_{c_2} * rand(\,) * (p_{gb} - p_{cc}) \tag{8}$$

$$v_a = w_a * v_a + a_1 * rand(\,) * (p_{ab} - p_a) + a_2 * rand(\,) * (p_{cb} - p_a) + a_3 * rand(\,) * (p_{cc} - p_a) \tag{9}$$

$$p_c = p_c + v_c \tag{10}$$

$$p_a = p_a + v_c + v_a\,. \tag{11}$$

Here, v_c is the velocity of a cluster center, p_{cc} is the position of the cluster center, p_{cb} is the position corresponding to the best fitness found so far for an agent within that cluster, p_{gb} is the global best, and w_c,

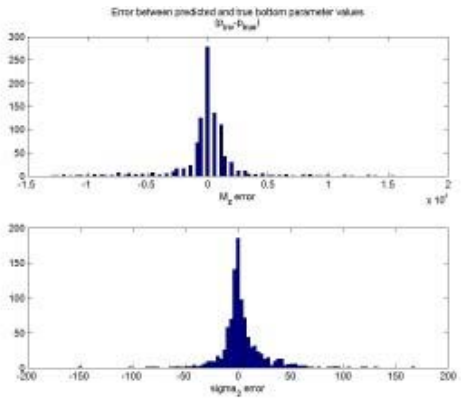

Figure 5 - Error in inverted bottom parameters compared to true bottom parameters based on exhaustive search of input space

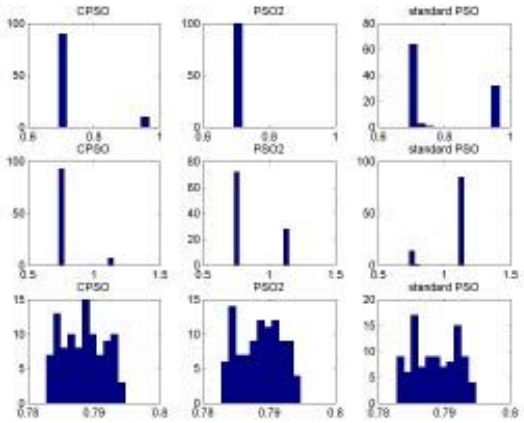

Figure 6 - Typical RMS error from inversion results over 200 realizations around a single operating point. Top -- PSO2 wins. Middle – CPSO wins. Bottom -- All three tie.

a_{c1}, a_{c2} are weighting factors for the cluster-velocity update equation. Additionally, v_a is the velocity of each agent, p_a is the position of an individual agent, p_{ab} is the personal best of each agent, and w_a, a_1, a_2, and a_3 are weighting factors for the agent-velocity update equation.

What we see then is the following: each of C total clusters has within it N agents, which are associated strictly with a single cluster only. First, the velocity of each cluster is updated as though the "center" of the cluster (effectively an arbitrary point in space, not the actual geometric center) was a standard PSO agent – it is modified based on its distance from some global best, along with its distance from its own personal best. Next, the velocity of each agent is calculated as a weighted sum of its distance from its own personal best, its cluster's best, and the cluster center, which serves to keep the cluster together. Finally, the position of the cluster center is adjusted by the center-velocity, and each agent is modified by both its cluster-center velocity and the individual agent-velocity update.

IV. Inversion Results

The model used to emulate the geoacoustic environment has thirty inputs: three sonar system parameters, and 27 environmental parameters including surface wind-speed, bathymetry, and the speed of sound at different depths in the water column. The three bottom parameters in which we are interested are M_z, σ_2, and ω_2. The grain size parameter M_z is a measure the log of the bulk grain size of the ocean floor sediment, and is used to categorize the floor as mud, sand, rock, etc. σ_2 is the ratio of the sediment volume scattering cross section to the sediment attenuation coefficient. It controls the sediment volume scattering strength in the model. Finally, ω_2 is the interface roughness spectral strength. A measure of the roughness of the ocean floor, it controls the interface roughness scattering strength in the model. While we would like to invert ω_2 along with the other parameters, for the purposes of testing the algorithm in this paper we will only invert the first two. The output of the neural network model consists of 599 neurons corresponding to 599 points in an underwater acoustic reverberation-level time series. This is the signal at an active sonar receiver due to scattering from inhomogeneities in the ocean medium, including the sea surface and the bottom. Figure 3 shows a typical example of this reverberation.

The neural network used to emulate the underwater acoustic model was trained using standard backpropagation methods [8] combined with a principal-component based method for reducing the redundant dimensionality of the data [9]. The resultant structure of the network consists of two hidden layers with 40 neurons in the first layer and 43 neurons in the second layer. With the fully trained neural network, we now have a method for calculating a fitness function for our optimizer. Specifically, we use the RMS error of the neural network output reverberation, as a function of the known inputs along with the current guess of the unknown parameters, compared to some target reverberation level obtained via the original underwater acoustic model (or a reverberation measurement obtained from an actual undersea environment).

As a test of the algorithm, we have set up the three PSO algorithms (standard PSO, CPSO, and PSO2) to invert only the two parameters M_z and σ_2. The parameters used in the latter two algorithms are listed in Table 1, along with a short description of the function of each parameter. We see in Figure 4 a

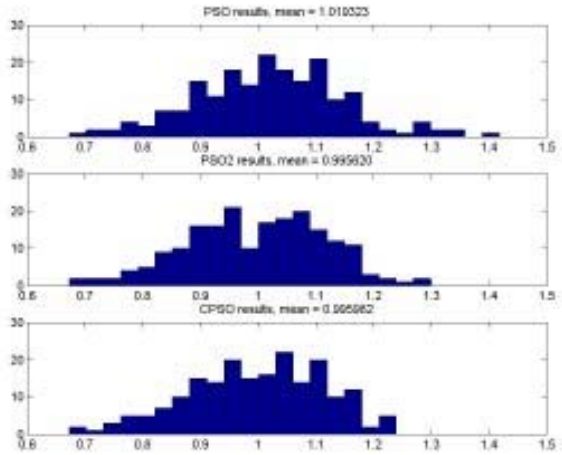

Figure 7 - Overall performance of each of the three algorithms in terms of their best RMS errors over the same 200 operating points.

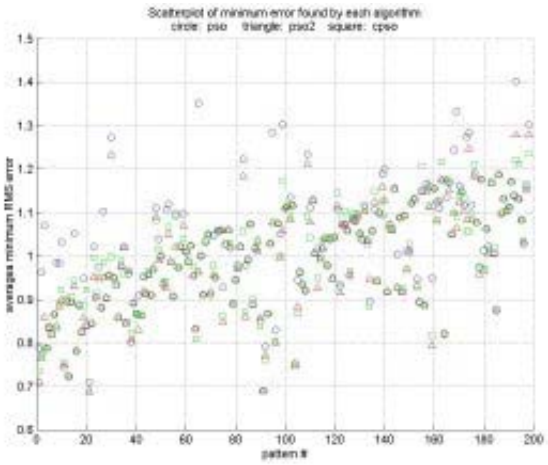

Figure 8 - Scatterplot of average RMS from each of 200 training patterns. Circle: PSO. Triangle: PSO2. Square: CPSO

typical error surface; we generated this error surface by first choosing an operating point (a specific input vector and corresponding target reverberation measurement), and then generating neural network outputs for every possible combination of M_z and σ_2, and calculating the corresponding error level. As described above, note the narrow valley-like area in which resides the global minimum, along with a fairly shallow local minimum far away from the global minimum. While Figure 4 does not describe *every* possible type of error surface (we have found a number with only a single bowl-like minimum as well), the shape represents a substantial portion of the error surfaces corresponding to each data point.

Parameter	***Description***	***Value***
PSO2		
δ_l	Long step-size	1
δ_s	Short step-size	0.1
CPSO		
w_c	Cluster inertia weight	0.9
a_{c1}	Cluster-best weight	2
a_{c2}	Global-best weight	2
w_a	Agent inertia weight	0.9
a_1	Agent-best weight	2
a_2	Agent Cluster-best weight	2
a_3	Agent Cluster-center weight	2

Table 1 – Parameter list for PSO methods

It should be noted at this point that any further discussion of inverting the neural network as described above to obtain the desired input parameters is worthless without first showing that the parameters obtained in such a manner actually correspond to the desired parameters. Since, for our model, we have a) the actual parameters used to generate a reverberation series, b) the target reverberation level itself, and c) a fully generated error surface in which the minimum is calculated simply by exhaustive search, we can quantify how well our inverted parameters match the actual true parameters, *under the assumption that the proposed algorithms worked perfectly*. Thus, we have a benchmark to which we can compare our PSO2 and CPSO results.

Figure 5 shows a histogram of the error between the inverted parameter value corresponding to the overall lowest error at a given operating point and the true parameter values used to calculate the target reverberation level. The top plot of this figure displays the errors in predicting M_z, and the bottom plot corresponds to errors in σ_2 prediction. We should note that the dynamic range of M_z is [-10000, 10000] (with values only landing on multiples of 500 – so the minimum possible nonzero error is +/- 500), and the dynamic range of σ_2 is all the integers on [1,200]. The data set from which the test cases were taken corresponds to the 1000 patterns which the neural network could reproduce the best (from the entire training data set). Thus, from these plots we can conclude that, in general, under the above assumption, our inverted parameters would match the desired parameters very well overall, with relatively few outliers. It should also be mentioned here that, even if the parameters do not match perfectly, what is needed in practice is the ability to generate a good reproduction of the ensonification of the water column

based on the inverted parameters, rather than specifically the parameters themselves.

We have quantified the performance of each algorithm by first running them at the same operating point (i.e., the same input vector and target reverb level) 100 times, and then generating a histogram of the minimum error found by the algorithm each time. Given the bimodal nature of most of the error surfaces (that is to say, there tends to exist a single global minimum and a single local minimum), we expect to see a histogram with a peak at either error level, or both. Figure 6 shows three typical plots of the results: one case in which CPSO outperforms PSO2, another in which PSO2 outperforms CPSO, and a third in which all three methods manage to find the global minimum quite well. It should be noted at this point that standard PSO almost never outperforms either of the other two algorithms.

Figure 7 shows another measure of performance of the three methods: histograms of the average RMS error found at each of 200 different operating points. We see from this plot that the overall performance, from the perspective of this figure of merit, is virtually identical for both PSO2 and CPSO, and that both algorithms do outperform standard PSO. Finally, Figure 8 shows a scatter plot of the lowest average error per pattern for each of the three methods. Again we see that both of the algorithms either outperform or match almost exactly with standard PSO in nearly every single case.

V. Conclusions and Further Research

We have shown that, for the purpose of inverting a neural network trained to emulate an underwater acoustic model in an attempt to determine certain bottom parameters, a particle-swarm optimization approach works very well. Moreover, both a two-step, gradient-approximation approach and a hierarchical cluster-based approach to modifying the standard PSO algorithm provide significant gains over the original algorithm. While these approaches were both tailored to the specific error surfaces to be optimized over in this problem, there exist many similar problems in which either of these approaches would be useful. Possible drawbacks exist, however. In the case of the two-step approach, twice as many fitness-function evaluations are required as in standard PSO. For the cluster method, Table 1 clearly illustrates a great deal of parameter tuning is necessary; even for this problem the listed values may be suboptimal.

As for further areas of research, first and foremost is the inversion of all three bottom parameters, including the ω_2 parameter. Likewise, we will soon work on a non-particle-swarm-based intelligent optimization scheme, such as simulated annealing or genetic algorithms as a method of comparison in inversion of the neural network. Finally, perhaps a fuzzy-systems approach to optimizing the many parameters in the CPSO approach (and perhaps even in PSO2) would offer even greater gain in performance.

VI. References

[1] Jensen, F. B., W. A. Kuperman, M. B. Porter, and H. Schmidt, *Computational Ocean Acoustics*. New York: Springer Verlag, 2000.

[2] Jensen, C.A.; Reed, R.D.; Marks, R.J., II; El-Sharkawi, M.A.; Jae-Byung Jung; Miyamoto, R.T.; Anderson, G.M.; Eggen, C.J., "Inversion of feedforward neural networks: algorithms and applications", *Proceedings of the IEEE*, Volume: 87 9, Sept. 1999 , pp. 1536 –1549

[3] Dosso, S., "Geoacoustic Inversion and Appraisal," OCEANS 2000 MTS/IEEE Conference and Exhibition, Volume: 1, 2000 pp. 423-428.

[4] Benson, J., Chapman, N.R., and Antoniou, A., "Geoacoustic Inversion with Artificial Neural Networks," OCEANS '99 MTS/IEEE. *Riding the Crest into the 21st Century*, Volume: 1, 1999, pp. 446-451.

[5] Fox, W. L. J., M. U. Hazen, C. J. Eggen, R. J. Marks II, and M. A. El-Sharkawi, "Environmentally adaptive sonar control in a tactical setting," in *Impact of Littoral Environmental Variability on Acoustic Predictions and Sonar Performance* (N. G. Pace and F. B. Jensen, eds.), pp. 595-602, Sept.2002.

[6] Eberhart, R., and Kennedy, J., "Particle Swarm Optimization," *1995 Proceedings of the IEEE Conference on Neural Networks*, Volume: 4, 1995. pp. 1942-1948.

[7] Eberhart, R., and Shi, Y., "A Modified Particle Swarm Optimizer," *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence*., The 1998 IEEE International Conference on , 4-9 May 1998 pp. 69 -73

[8] Reed, R. D., and Marks, R. J. II, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks,* MIT Press, Cambridge, MA, 1999.

[9] Mann, T., *et al.,* "Preprocessing Neural Network Outputs For Improved Tolerance To Error," 2003 Proceedings of the IEEE International Joint Conference on Neural Networks, forthcoming.