ANL/IPNS/CP-98087

ICANS-XIV
14th Meeting of the International Collaboration on
Advance Neutron Sources
June 14–19, 1998
Utica, IL USA

Neutron Production Enhancements for the Intense Pulsed Neutron Source

E. B. Iverson, J. M. Carpenter, T. L. Scott, M. E. Miller, L. D. Peters, M. W. Schlueter
Intense Pulsed Neutron Source
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL USA 60439

The Intense Pulsed Neutron Source (IPNS) was the first high energy spallation neutron source in the United States dedicated to materials research. It has operated for sixteen years, and in that time has had a very prolific record concerning the development of new target and moderator systems for pulsed spallation sources. IPNS supports a very productive user program on its thirteen instruments, which are oversubscribed by more than two times, meanwhile having an excellent overall reliability of 95%. Although the proton beam power is relatively low at 7 kW, the target and moderator systems are very efficient. The typical beam power which gives an equivalent flux for long-wavelength neutrons is about 60 kW, due to the use of a uranium target and liquid and solid methane moderators, precluded at some sources due to a higher accelerator power.

The development of new target and moderator systems is by no means stagnant at IPNS. We are presently considering numerous enhancements to the target and moderators that offer prospects for increasing the useful neutron production by substantial factors. Many of these enhancements could be combined, although their combined benefit has not yet been well established. Meanwhile, IPNS is embarking on a coherent program of study concerning these improvements and their possible combination and implementation. Moreover, any improvements accomplished at IPNS would immediately increase the performance of IPNS instruments.

1 Enhancements In Progress

A number of enhancements to IPNS neutron production are already in progress. Design choices for these enhancements have largely been made, and engineering questions are all that remain to be answered, whether by experiment or calculation. These enhancements are relatively well-defined, with clearly demonstrable benefits. They include a re-designed booster target, based on experience with the booster target used from 1988–1991, re-designed solid methane moderators for use with the booster target, a moderator-reflector assembly designed for rapid moderator replacement, and experimental studies of minor moderator modifications.

1.1 New Booster Target

IPNS has operated with a depleted uranium target from 1981–1988 and since 1992. For three years, 1988–1991, we operated with a booster target composed of the same alloy of α-phase uranium enriched to 77% ^{235}U, and of the same physical design as the depleted target. This subcritical target had a multiplication factor k_{eff} of approximately 0.80, and resulted in neutron production of about two and one-half times the production rate with the depleted uranium target.

Nonlinear Programming with Feedforward Neural Networks

Jaques Reifman
Argonne National Laboratory
Reactor Analysis Division
9700 S. Cass Avenue
jreifman@anl.gov

Earl E. Feldman
Argonne National Laboratory
Reactor Analysis Division
9700 S. Cass Avenue
feldman@ra.anl.gov

Abstract

We provide a practical and effective method for solving constrained optimization problems by successively training a multilayer feedforward neural network in a coupled neural-network/objective-function representation. Nonlinear programming problems are easily mapped into this representation which has a simpler and more transparent method of solution than optimization performed with Hopfield-like networks and poses very mild requirements on the functions appearing in the problem. Simulation results are illustrated and compared with an off-the-shelf optimization tool.

Introduction

The Hopfield network and variations of this type of neural networks have been frequently considered as candidates for solving optimization problems, such as combinatorial optimization [1], linear programming problems [2], and nonlinear programming problems [3]. This is usually achieved by designing Hopfield-like networks whose energy function mimics a cost function which embodies the optimization problem to be solved. Hence, the solution to the optimization problem is obtained by attaining the lowest energy state of the network.

Here, we propose a new method based on the widely used multilayer feedforward neural networks (FNNs), also known as the multilayer perceptron, for solving nonlinear mathematical programming problems. The method of solution for the proposed approach is simpler and more transparent than the existing Hopfield-like approaches and unlike the method in [3] poses very mild requirements on the functions appearing in the problem.

The Nonlinear Programming Problem

Consider the general mathematical program (P)[1] of the form:

$$\text{minimize } f(x) \tag{1}$$

subject to inequality and equality constraints,

$$g_p(x) \geq 0, \qquad p=1,2,\ldots,P \tag{2}$$

$$h_q(x) = 0, \qquad q=1,2,\ldots,Q \tag{3}$$

where the objective function f to be minimized and the constraints $g_1,\ldots,g_P$, and $h_1,\ldots,h_Q$, can be linear or nonlinear functions of the N-dimensional vector $\boldsymbol{x}$.

A vector $\boldsymbol{x}$ is called a *feasible solution* to (P) iff $\boldsymbol{x}$ satisfies the P+Q constraints of (P). The collection of such $\boldsymbol{x}$ is called the *feasible set* and $\boldsymbol{x}^*$ is the feasible solution which yields the minimum f, i.e., the solution to (P).

The Neural Network Formulation

In the proposed FNN formulation, the solution to (P) is obtained by transforming this *constrained* optimization program into a series of *unconstrained* optimization programs (P'), Eq. (4), which are solved for a sequence of f_k, (k=1,2,...,K). In Eq. (4), w is the M-dimensional weight vector of a FNN where $\boldsymbol{x}$ is a function of w, f_k is a selected feasible value of f, and ρ is a positive number used to vary the weight of the constraint terms. At each optimization we seek the vector $\boldsymbol{x}_k^*$ which minimizes $F_k(w)$ for a given f_k

[1](P) and other capital letters when enclosed in parentheses represent mathematical programs.

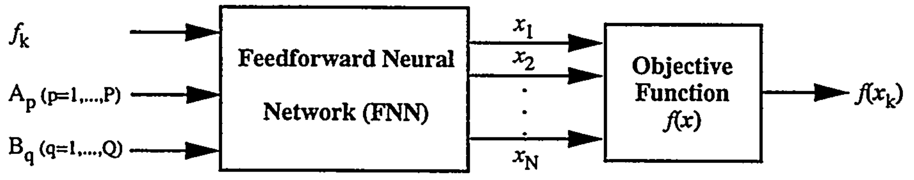

Fig. 1. Coupled neural-network/objective-function representation

such that at the end of the sequence of optimizations $x^*_K \approx x^*$. That is, the solution for (P') at k=K is the solution for (P). Through this procedure we iterate on f and try to find its smallest value which has feasible x.

$$\text{minimize} \left\{ F_k(w) = \frac{1}{2}\{f_k - f(x)\}^2 + \frac{\rho}{2}\sum_{q=1}^{Q}\{h_q(x)\}^2 + \frac{\rho}{2}\sum_{p=1}^{P}\{\min[0, g_p(x)]\}^2 \right\} \tag{4}$$

The first term on the right side of Eq. (4) assures that x^*_k yields the desired f_k, the second term accounts for the equality constraints and the third term accounts for the inequality constraints. Note that if x satisfies an inequality constraint of (P), the corresponding element of the third term becomes zero and if x is a feasible solution to (P), then the second and third terms become zero.

This formulation is somewhat of a combination of the *exterior penalty function method* [4] with the *least-squares method* [5]. As in the exterior penalty function method, for each equality constraint and for each inequality constraint (except for two-sided bounding inequality constraints of single variables, which are directly accounted for as described later) of the program (P), there is a corresponding term in the objective function F_k of program (P'). However, unlike the penalty function method, the first term of F_k consists of the square of the difference $\{f_k - f(x)\}$, as opposed to just $f(x)$, and the minimum of F_k is zero. In this sense, the form of F_k is similar to the objective function of the least-squares method which minimizes the squares of the differences between the observed values and their respective predicted values.

Here we cast the minimization of $F_k(w)$ for each f_k as the training of the FNN in the coupled neural-network/objective-function representation illustrated in Fig. 1. In this representation, the left box consists of a FNN with M weights w and the right box consists of the objective function $f(x)$ of (P), which perhaps could be represented by a trained neural network. Training consists of determining the weights w that for a given input to the FNN the network provides outputs x_n, (n=1,...,N), which minimize $F_k(w)$ in Eq. (4).

In addition to f_k, the other inputs to the FNN are the constant terms A_p and B_q that appear in the constraints g_p, (p=1,...,P), and h_q, (q=1,...,Q). For example, for the equality constraint $x^3+x-5=0$, the number 5 would be used as an input. If a constraint does not have a constant term, then zero should be used as input such that there are a total of P+Q+1 inputs to the FNN. In theory, this set of inputs is not necessary because they already appear in Eq. (4). However, simulation results indicate that inputing these values instead of dummy values improves training.

Because each one of the N output nodes of the FNN corresponds to one independent variable x_n, (n=1,...,N), two-sided bounding inequality constrains on each variable x_n can be directly treated through the proper selection of the mapping function of the output nodes and proper normalization. For example, if $a \le x_n \le b$, where a and b are scalars, then the *n-th* output node could be mapped by a sigmoid function with the range [a,b] normalized to the range [0,1] of the sigmoid function. Each two-sided constraint could, of course, be separated into two one-sided constraints and treated in the third term in Eq. (4). This would require, however, the addition of two more elements per two-sided constraint. The FNN formulation eliminates this need and when these are the only types of constraints in the problem the last two terms in Eq. (4) vanish, greatly simplifying the minimization of F_k.

In this neural network formulation, the solution to (P) is obtained as follows. We start by selecting an interval $f_{low} \le f(x^*) \le f_{up}$ that includes the solution to (P). In most engineering problems at least one of the bounds of the interval is directly obtained from the physical constraints on the solution. Then, we start to solve the unconstrained optimization in Eq. (4) for a sequence of $f_1,...,f_k...,f_K$, such that at each optimization step of the sequence f_k is selected to reduce by half the interval $[f_{low}, f_{up}]$ according to the bisection method [6]. The sequence proceeds until the interval has been reduced to within a prespecified distance ε, with $f(x^*)$ contained in the interval.

For each optimization step of the sequence we train the FNN in Fig. 1 with the selected f_k and the constants A_p, (p=1,...P), and B_q, (q=1,...,Q), which are kept fixed for the entire sequence. If the training is successful, i.e., if weights w can be found that minimize $F_k(w)$ for the current f_k, then x_k^*, provided as the output of the FNN, is a feasible solution to (P). If the training is not successful we select another value for f_k according to the bisection method and continue the procedure. By repeating this procedure for additional value of f_k we obtain the smallest f_k, $f_{k=K}$, within a prespecified tolerance, for which the training of the FNN converges. Corresponding to f_K, we obtain $x_K^* \approx x^*$, the solution to (P). This can be confirmed by showing that x_K^* satisfies the Karush-Kuhn-Tucker necessary conditions for local optimality of nonlinear constrained functions to within a certain tolerance [7].

Our method differs from most nonlinear programming approaches because here we solve the inverse problem. That is, we select a value of $f(x)$, f_k, and try to obtain the corresponding feasible x, if it exits. Hence, our iterative procedure is based on a search for the smallest $f(x)$ with feasible x, $f(x^*)$, along the monotonicly decreasing f line, as opposed to a direct search for x^* in the N-dimensional x-space. This precludes the search from becoming trapped at a local minimum in the x-space. However, the approach is not completely free of local minima trapping because the training of the FNN may not converge even when $f_k > f_K \approx f(x^*)$. Therefore, whenever a training session is not successful the network should be retrained with the same inputs but with a different selection of the initial weights and a different number of nodes in the hidden layers to ensure that the unsuccessful training is not due to local minima in the w-space.

In that sense, we might be transferring the potential trapping in the N-dimensional x-space to a potential trapping in the M-dimensional w-space of the network weights. The search in the N-dimensional x-space is constrained while the search in the M-dimensional w-space is not, although, in general, M>>N.

The use of the FNN in our formulation is also quite different from its common use. Instead of providing a set of input-output pairs and having the network learn their underlying relationships, we provide only inputs and for each training session the same input is presented repeatedly to the network. We are not interested in the generalization capabilities of FNNs, but rather are using the neural network representation in Fig. 1 to minimize F_k.

The proposed approach poses very mild requirements on the functions appearing in the problem. We assume that the nonlinear programming problem has the following properties:

1. The functions $f(x)$, $g_p(x)$, and $h_q(x)$ all have continuous first derivatives.

2. $f(x)$ is continuous in an interval of non-zero length $f(x^*) \le f(x) \le f_{up}$ and there must be a corresponding feasible x for each value of $f(x)$ in the interval.

3. $N \ge Q + 1$, as opposed to $N \ge Q$ for conventional methods, because an extra degree of freedom is needed for the f_k search.

Neural Network Training

The unconstrained minimization of $F_k(w)$ in Eq. (4) is solved iteratively based on calculations of the gradient $\nabla F_k(w)$ using a conjugate gradient version of the backpropagation algorithm [8]. The method of conjugate gradient expedites the training process and dynamically optimizes the learning parameter and the momentum parameter of the backpropagation algorithm.

As in other versions of the backpropagation algorithm, the components of $\nabla F_k(w)$ are computed recursively by starting at the nodes in the output layer of the FNN and working backward to the nodes in the input layer. A component of $\nabla F_k(w)$ corresponding to the weight $w_{ji}^{(\ell)}$ connecting the *i-th* node in the *(ℓ-1)-th* layer to the *j-th* node in the *ℓ-th* layer is given by

$$\frac{\partial F_k}{\partial w_{ji}^{(\ell)}} = -\left[\delta_j^{(\ell)} + \sum_{p=1}^{P} \delta_{pj}^{(\ell)} + \sum_{q=1}^{Q} \delta_{qj}^{(\ell)}\right] y_i^{(\ell-1)}, \qquad (5)$$

where $y_i^{(\ell-1)}$ denotes the activation of the *i-th* node in the *(ℓ-1)-th* layer and to simplify the notation we suppress the subscript k in the δs and y. If the *ℓ-th* layer is the output layer L, then

$$\begin{aligned} \delta_j^{(L)} &= \left[f_k - f(x_k)\right] y_j^{(L)} (1 - y_j^{(L)})\, f_j'(x_k) \\ \delta_{pj}^{(L)} &= \begin{cases} 0 & ; \textit{if } g_p(x_k) \ge 0 \\ \rho\, g_p(x_k)\, y_j^{(L)} (1 - y_j^{(L)}) & ; \text{otherwise} \end{cases} \\ \delta_{qj}^{(L)} &= \rho\, h_q(x_k)\, y_j^{(L)} (1 - y_j^{(L)}) \end{aligned} \qquad (6)$$

where the N elements of x_k are equal to the N elements of $y_j^{(L)}$ and $f_j'(x_k)$ is the partial derivative of $f(x)$ with respect to the *j-th* element of x (or the *j-th* output node of the FNN, $y_j^{(L)}$). If the function $f(x)$ is represented by a multilayer FNN, then $f_j'(x_k)$ is the partial derivative of this network's output with respect to its inputs. For any node in a subsequent hidden layer, i.e., $1<\ell<L$,

$$\begin{aligned}
\delta_j^{(\ell)} &= y_j^{(\ell)}(1-y_j^{(\ell)})\sum_{m=1}^{J_{\ell+1}} \delta_{jm}^{(\ell+1)}\, w_{mj}^{(\ell+1)},\\
\delta_{pj}^{(\ell)} &= y_j^{(\ell)}(1-y_j^{(\ell)})\sum_{m=1}^{J_{\ell+1}} \delta_{pm}^{(\ell+1)}\, w_{mj}^{(\ell+1)}, \qquad (7)\\
\delta_{qj}^{(\ell)} &= y_j^{(\ell)}(1-y_j^{(\ell)})\sum_{m=1}^{J_{\ell+1}} \delta_{qm}^{(\ell+1)}\, w_{mj}^{(\ell+1)},
\end{aligned}$$

where $J_{\ell+1}$ denotes the number of nodes in the $(\ell+1)$*-th* layer.

This algorithm is very similar to the standard version of the backpropagation algorithm used to compute $\partial F/\partial w_{ji}^{(\ell)}$ for stand-alone FNNs. The major differences are the presence of three δs, as opposed to only one, corresponding to the three terms in Eq. (4), and the extra term $f_j'(x_k)$ in Eq. (6).

Simulations

To demonstrate the performance of the proposed FNN approach, some examples of an emissions control problem [9] with different types of constraints are presented. The problem consists of finding the optimal flow rate of four natural gas injectors x_n, (n=1,2,3,4), located above the primary combustion zone of a coal-fired power plant such that the downstream oxides of nitrogen (NO_x) emissions are minimized. A trained multilayer FNN is used to represent the nonlinear functional relationship $f(x)$ between the injected gas and the NO_x emissions.

A. *Inequality Constraints*

Consider the following nonlinear program:

$$\text{minimize } \{NO_x = f(x_1,x_2,x_3,x_4)\} \qquad (8)$$

subject to

$$34.90 \le x_n \le 72.12, \quad (n=1,2,3,4) \qquad (9)$$

$$110 - \sum_{n=1}^{3} x_n \ge 0. \qquad (10)$$

A three-layer FNN with a 2-6-4 architecture was used. The two nodes in the input layer correspond to a selected value for NO_x and 110, respectively, and the four nodes in the output layer correspond to the gas flow rate of the four injectors.

Table 1 compares the solutions to this program obtained by the FNN method and an off-the-shelf optimization tool [10]. The deviation of the optimal NO_x was less than 0.3% and the maximum deviation of x_n was less than 2.0%. This is a rather difficult optimization problem for some methods because the optimal solution is on the boundary of the feasible region prescribed by Eq. (10). Similar comparisons were obtained in other simulations where we changed the sign of the inequality and the value of the constant term in Eq. (10).

Table 1. Comparison of the FNN results with an off-the-shelf optimization tool for a case of inequality constraints

	NO_x	x_1	x_2	x_3	x_4
FNN	0.415	35.46	35.41	39.04	71.85
Tool	0.414	35.26	34.90	39.81	72.12

B. *Equality and Inequality Constraints*

Consider next the same program above with the inequality constraint in Eq. (10) replaced by the equality constraint

$$175 - \sum_{n=1}^{N=4} x_n = 0. \qquad (11)$$

Employing the same FNN architecture as in the previous case and with the constant term in the constraint, 175, used as an input we obtained the results illustrated in Table 2. The FNN results compare well with results obtained with an off-the-shelf optimization tool. Both attained the same minimum value for NO_x and the maximum deviation on x_n was less than 1%. Similar comparisons were obtained in other simulations where we changed the value of the constant term in Eq. (11).

Table 2. Comparison of the FNN results with an off-the-shelf optimization tool for a case of equality and inequality constraints

	NO_x	x_1	x_2	x_3	x_4
FNN	0.431	34.95	34.98	35.23	69.89
Tool	0.431	34.90	34.90	34.90	70.30

Conclusions

We propose a new methodology for solving nonlinear programming problems. The approach is to transform an original constrained optimization problem in the N-dimensional x-space into a sequence of unconstrained optimization problems in a larger M-dimensional weight-space of a multilayer feedforward neural network. Although M>>N, the difficulty in solving an optimization problem in the larger weight space is more than offset by the simplicity of solving an unconstrained optimization problem, as opposed to a constrained one, in the smaller x-space.

The constraints of the original problem are handled indirectly through the transformation of the original function $f(x)$ into a modified function which incorporates each equality constraint and each inequality constraint into an additional term of the function. Two-sided bounding inequality constraints of single variables are directly treated through proper selection of the mapping function of output nodes of the neural network and proper normalization.

In contrast to most optimization approaches, the optimal solution is not obtained through searches in the x-space. Instead, we directly search for the smallest $f(x)$ along the monotonic $f(x)$ line and indirectly obtain the corresponding x (as the output of a feedforward neural network) by training the network. Hence, the method of solution is not dependent on the form of $f(x)$ on x, and therefore, should be less sensitive to local minima, and poses very mild requirements on the functions appearing in the problem.

The examples provided serve to illustrate that the results of the proposed method compare well with those of other optimization techniques. In future research we shall demonstrate the capability of the proposed method to converge to the global minimum even when the $f(x)$ surface contains many local minima. In general, the existence of these local minima are problematic for optimization methods.

The proposed method should also be quite appealing in problems involving two-sided bounding inequality constraints on single variables. Unlike most approaches, these constraints are directly satisfied through the proper selection of the mapping function of the network output nodes and proper normalization.

The proposed method is simpler and more transparent than existing neural network approaches for solving nonlinear programming problems. However, it requires the solution of a sequence of optimizations. This iterative procedure will be avoided in future research by modifying the first term in Eq. (4), $[f_k - f(x)]$, to $f(x)$, and directly solving the unconstrained optimization problem much like the exterior penalty function method.

Acknowledgements

This project was supported by the U.S. Department of Energy, Office of Science, Small Business Technology Transfer Research and the Laboratory Directed Research and Development program at Argonne National Laboratory.

References

[1] J. J. Hopfield and D. W. Tank, "Neural Computation of Decisions in Optimization Problems," *Biol. Cybern.*, vol. 52, pp. 141-152, 1985.

[2] D. W. Tank and J. J. Hopfield, "Simple Neural Optimization Networks: an A/D Converter, Signal Decision Network, and a Linear Programming Circuit," *IEEE Trans. Circuits Syst.*, vol. CAS-33, pp. 533-541, May 1986.

[3] C.-Y. Mass and M. A. Shanblatt, "A Two-Phase Optimization Neural Network," *IEEE Trans. Neural Networks*, vol. 3, pp. 1003-1009, 1992.

[4] M. Avriel, *Nonlinear Programming: Analysis and Methods*, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[5] W. Spendley, "Nonlinear Least Squares Fitting Using a Modified Simplex Minimization Method," *Optimization*, Eds. R. Fletcher and A. E. R. E. Harwell, Academic Press, New York, New York, 1969.

[6] R. W. Hornbeck, *Numerical Methods*, Quantum Publishers, New York, New York, 1975.

[7] P. Y. Papalambros and D. J. Wilde, *Principles of Optimal Design*, Cambridge University Press, New York, New York, 1988.

[8] J. Reifman and J. E. Vitela, "Accelerating Learning of Neural Networks With Conjugate Gradients for Nuclear Power Plant Applications," *Nucl. Technol.*, vol. 106, pp. 225-241, 1994.

[9] J. Reifman, E. E. Feldman, T. Y. C. Wei, R. W. Glickert, J. M. Pratapas, and J. S. Herzau, "Neural Network Optimization of Boiler Emissions Control With Gas Reburn," paper presented at the AIChE 1999 Spring National Meeting, Houston, Texas, 14-18 March, 1999.

[10] Solver, Frontline Systems, Inc., Incline Village, NV, 89450.