# System Identification Using Off-Optimum Data From A Genetic Algorithm: A Source Of Training Data For an Artificial Neural Network

Stephen Shervais
Assistant Professor, MIS
College of Business and Public Administration
Eastern Washington University
Cheney, WA, 99004
509.359.4280
sshervais@ewu.edu

**Abstract.** When developing an artificial neural net model of a system, the most efficient way to obtain training and test data is often to generate a large set of random inputs and run them through the model. But that is not the only way to do it. We demonstrate the use of genetic algorithm-generated data as a source of input-output pairs for training an artificial neural network. If the genetic algorithm and neural network are being developed together – for example, to provide system identification in support of a control system – this data is readily available and performs as well as a random search of the state space.

## Categories and Subject Descriptors
G.1.6 [**Optimization**]: Global optimization

## General Terms
Algorithms

## Keywords
Genetic algorithms, artificial neural networks

## 1. Introduction
A seldom-noted fact about the operation of a genetic algorithm (GA) is that the majority of the information generated by the GA is discarded as part of the normal operation of the evolutionary process. In cases where it *is* used, the goal is usually to improve the operation of the GA itself [12], [11]. This is understandable. In the majority of cases, the object is to find the single best set of inputs to a system in order to obtain an optimum output. But the discarded data, while not particularly useful in describing the optimum of the system, can be used to describe the response of the system to an arbitrary input – that is, to perform system identification.

The fitness function of a GA is, of course, one kind of model of the system under consideration. However, in some cases the fitness function model is not appropriate for use elsewhere, particularly in a control environment. For example, Wu [24] proposed using an artificial neural network (NN) as an adjunct to a supply chain control system to compensate for the fact that information on the system's states might be delayed. In another case [20], the requirement calls for a differentiable model, and the fitness function, as written, might not be. Finally, the response time of the fitness function might not be quick enough for the control requirements, particularly if it is a discrete event simulation.
One possible approach to these problems is to create an artificial neural network (NN) and train it as a quick-reacting, differentiable model of the system in question. To do this, we require a set of input-output (I/O) pairs to use as training data. Usually, the most efficient way to do this is to generate inputs at random across the state space, and pass them through the mathematical model of the system. This approach is both efficient, because there is no additional processing overhead, and effective, because it satisfies the NN requirement for inputs that span the full space of the system. But just because it is efficient and effective does not mean that it's the only source of data. If, for example, the NN is being developed in conjunction with a GA – the GA finding the optimum state of the system the NN is modeling – then the GA-generated data is available, essentially for free, and can also be used to train the NN.

This paper tests two different ways of obtaining that data: from a GA, and from a random sample of the input space. It does this in the context of two separate, but related, questions. First, can GA-derived data train a NN as well as can randomly generated data? Second, can a NN controller that uses a GA-based NN system model perform as well as a controller that uses a NN model based on random data. In Section 2, we discuss the use of NNs in the development and operation of control systems, with emphasis on adaptive critic controllers. Section 3 takes up the issue of using GA-derived data to train the NNs. In Section 4 we lay out some simple test models, and describe the methodology. Section 5 discusses the results.

## 2. Neural Control Systems
An artificial neural net (NN) is a multi-layer network of non-linear processing elements linked by weighted paths. In operation, the values of the interlayer weights are first initialized

to random values, and the NN is trained to produce a specific output (O) from a given input (I). The training process may be thought of as a multi-step process generating a sequence of improved approximate models based on exposure to a series of I/O pairs.

The need for adaptation is one long recognized in the controls community and is driven by our inability to precisely specify the characteristics of the system to be controlled and the tendency for the parameters of such systems to vary over time [1]. The use of artificial neural nets as adaptive, optimizing controllers was prompted by the work done by Werbos [20], [21], and Wunsch [25], but early work on adaptive control may be tracked back to [23] Widrow. A good survey was prepared by Werbos [22].

Prokhorov and Wunsch [14], [15], [25], have been actively developing a family of adaptive critic NN designs based on ideas originally suggested by Werbos. One such design is called Dual Heuristic Programming (DHP), and its goal is to adjust the design of a controller by estimating Bellman's [2], [3], [4], [5] dynamic programming J-function. In DHP, the controller is adjusted based on predictions made by a differentiable model of the system to be controlled. "The use of derivatives of an optimization criterion, rather than the optimization criterion itself, is known as being the most important information to have in order to find an optimal solution." ([25], pg. 104).

Artificial neural nets are a particularly useful modeling tool for system such as inventory control and reordering systems because in spite of the fact that the cost equation is linear, the system response certainly is not, and the sigmoid function in a NN processing element can approximate a hard-limit function of the form $f(\mathrm{x}) = \left\{ \begin{matrix} 0 \text{ if } \mathrm{x} \le 0 \\ 1 \text{ otherwise} \end{matrix} \right\}$, which is exactly the situation we encounter when a stock purchase might or might not occur, depending on the relationship between the stock level and the reorder point [16].

So, a NN provides a differentiable model, and a NN is trained through the use of input/output pairs from the system to be modeled. A key question, of course, is where do the I/O pairs that describe this relationship come from? Requirements for the data are that they span the expected operating region of the system, that they provide enough samples of critical regions that the NN is not overwhelmed by the (possibly more common) non-critical regions, and that they be different enough to satisfy the control engineer's requirement for 'persistence of excitation' to the controller. There are four possible ways to generate I/O pairs for use in training a system model NN:

1. Measure the inputs and outputs of a real-world system. This has the advantage of authenticity, but has the drawbacks that it may not sample the entire operating space, might not provide the necessary excitation, and – if the system has not been built – may not exist.
2. Make a random selection of inputs and use a simulator to generate the appropriate outputs. This will provide the span and excitation, but it might not provide the data needed from the critical regions.
3. Step across the input state space, using a defined sampling strategy and a simulator to generate the appropriate outputs. This will provide the span and excitation, but the ability This will provide the span and excitation, but the ability to find critical regions is dependent upon the grid size.
4. Use I/O pairs generated by a genetic algorithm process (see next section). Such data has the advantage of being a recyclable by-product of a GA search for an optimal system. Another advantage of off-optimal data is that it is plentiful. A 100 population, 10,000 generation GA will produce on the order of $10^6$ I/O pairs. In addition, if the input variables don't change, and the fitness function remains the same, data from any preliminary runs is available as well. This advantage is tempered by the fact that many of the I/O pairs will be identical from generation to generation, and some process is needed to retrieve unique training records. Such winnowing processes add to the computational overhead. Another theoretical advantage, from a controls standpoint, is the fact that a GA spends much of its time (in the later generations, at least) in the vicinity of the system optimum. This means the area around the optimum is better mapped than areas well away from it. On the other hand, it means that areas away from the optimum will be less well mapped. The desirability of this situation will depend on the unique circumstances of the system being controlled.

This paper limits its consideration to options 2 and 4.

## 3. Genetic Algorithms and Neural Net Training.

This section describes prior work in the area. A number of researchers, discussed below, have used GAs as sources of training information for NNs to help solve industrial problems. It is interesting to note that in none of these approaches do the researchers use off-optimal data from the GA as a source of training information for the NN.

### 3.1 Ishigami

Ishigami and his co-workers [9] report on a system used to control voltage stability in electric power networks, a continuous system with discrete control settings. There, the GA starts with a representation of the distributed control elements available to a power grid and works to limit the voltage fluctuations under different demand regimes. When the GA arrives at a solution, that information is used to generate the training set for a NN, which is to operate the control equipment. In a similar effort, Pipe [13] used a GA to train an Adaptive Heuristic Critic NN to learn a maze. The GA generated suggested movements, and the AHC's V-function was used as the fitness function for the GA.

### 3.2 Ventura and Martinez

Ventura and Martinez studied the use of genetic algorithms as a source of training data for neural control systems. While some of their work has included discrete values, they did not use a discrete event system as such.

Their original work [17] used a GA to develop optimal solutions to a set of randomly generated Boolean equations, and used the equation/solution pairs as the training set for a neural net. They then moved on to develop pointing solutions for a gun/target problem [18], [19]. Given the target initial coordinates and vector, the GA found an optimal aim point to hit the target. The coordinate, vector, and aim point data then were used to train the NN.

## 3.3 Caskey and Storch

Caskey and Storch [6], [7], have worked on optimizing the configuration of discrete event systems, specifically machine shops. Their work is at a slightly higher level of abstraction than most, because they included the shop type -- job, flow, or hybrid -- as one of the input variables, and they developed a "one-shot" optimizing tool. Their inputs are major policy elements such as shop type, due-date assignment rules, and tardiness policies. For each set, the GA develops an optimal scheduling rule for each machine. They use a discrete event simulator to rank the GA solutions. The C&S approach limits the use of the NN to that of a rule repository. Of all the machine scheduling problems, the flow shop comes closest to matching the inventory control problem, and some of their results may be applicable here.

## 3.4 Lin and Jou

Lin and Jou [10] have applied fuzzy-neuro-genetic adaptive critic techniques to training a controller for chaotic systems. In their approach, the GA generates a set of candidate Action Nets by encoding sets of weights. Then the system runs for some short time period, during which the update signal from the Critic Net is used as the fitness function, or the chaotic system makes an unacceptable departure from the desired track. This means that Action Nets that run for longer times without failure have greater fitness. They report success in applying this technique to control of two benchmark chaotic systems, the Henon map and the logistic map.

## 4. Methodology

As discussed in Section 2, a training data set for a NN must span the region of interest, provide significant coverage of critical areas, and maintain persistence of excitation. One of the objects of this research was determining which of two possible approaches to the problem – random or genetic algorithm – provided a training set that would create the best NN system model.

## 4.1 The Simulation

The fitness function for these experiments was a discrete event inventory control and transportation simulation that had been developed to work with a GA in the design of adaptive critic controllers [16]. At each timestep the simulation generated 36 state variables describing stock on hand and in transit, 18 control inputs (the variables being optimized by the GA) specifying the operating policies (reorder point RP, order up-to point UT, and transport capacity procured $T_{cap}$ (collectively designated $u(t)$). In addition, there were 4 exogenous, non-stationary, stochastic inputs $Q_D(t)$ representing retail demand. All of these were indexed for the specific node **n**, stock **k**, transport arc **a** or transport mode **m** involved. The stochastic demand variables remove stock quantities from the demand nodes. When a stock level at a node is reduced to or below the reorder point policy value for that stock at that node RP(n,k,t), an order is placed for resupply from the next node up the supply chain. The order quantity is determined by the order-up-to quantity policy value UT(n,k,t). When the order is filled, it is shipped subject to the available capacity of the intervening transit arcs as determined by the transit capacity policy values $T_{cap}(a,m,t)$.

The function to be minimized is total cost CT, which consists of the linear combination of initial (Co) and final (Cx) costs, plus the incremental costs (holding cost, Ch; purchase cost, Cq; transport cost, Ct; and stockout penalties, Cp) at each timestep (t), summed over N nodes, K stocks, and the planning time horizon T:

$$C_T = C_0 + \sum_{n=0}^{N} \sum_{k=0}^{K} \sum_{t=0}^{T} \left( Ch_{(t)} + Cq_{(t)} + Ct_{(t)} + Cp_{(t)} \right) + Cx.$$

Within each of these cost elements, there were a number of step functions, that made the cost functions discontinuous. For example:

$$X_{nk(t)} = \begin{cases} Qu_{nk} - \left( Q_{nk(t)} + Xe_{nk(t)} \right) \text{ if } Q_{nk(t)} < Qr_{nk(t)}, \\ \text{otherwise, } 0, \end{cases}$$

determines the size of the resupply order for stock n at node k and time t, where the Q terms refer to the GA-adjusted rules for reordering, and the Xe term refers to stock enroute.

The demand schedule for the GA optimization process, $Q_D(t)$, was Poisson distribution-derived, with stationary mean and variance, and covered a period of 90 days. The demand schedule for the controller training process was identical, but three different test schedules were used, each one was 360 days long – there was a new set of stationary stochastic data (*Baseline*), a set with a step in demand, followed by a stationary process at the new mean (*Stepped Demand*), and a schedule with constantly increasing demand across the 360 days (*Increasing Demand*)

## 4.2 The Genetic Algorithm.

The GA implemented for the present work used a chromosome with three genes for each node – initial stock level, reorder point, and order-up-to point. It also had one gene for each transport arc and mode – transport available.

In this GA, the top half of the population reproduces stochastically (roulette wheel selection), with reproduction probability based on sigma scaling [8]. Crossover is a version of uniform crossover. The probability of crossover was set at 0.80, and the mutation rate was set at 0.01. Business constraints necessary to the operation of the simulation are handled by repairing the chromosome as it is being created.

## 4.3 The Training Data

The two data sets were created as follows:

**Genetic Algorithm:** A 90-day run on the simulator was used to create the I/O pairs. This provided an embarrassment of riches. If the GA was run for 1000 generations with a population of 100, this could result in some nine million data pairs, or about 1.75GB of data, far too much data for a realistic training regime. The procedure used was to save the simulation outputs only once every 97 generations. This produced files that were about 65 to 70MB and contained 290,000 to 360,000 records. Record numbers changed from run to run, because during a given generation the GA would only run policy sets it hadn't looked at previously, and the number of new policy sets depended on the survival rates of earlier solutions.

**Random:** The GA code was used, but the population was initialized every generation with random individuals. The number of generations was reduced, but all data outputs were kept. The generations was reduced, but all data outputs were kept. The data runs were still 90 days long.

## 4.4 The Tests

Each system model NN was trained using one of these data sets. Training parameters were: learning coefficient 0.98, training epoch size 16, with one training session involving ten looks at the full data set. At the end of the training the NN was tested on the holdout data. One hundred reinitializations were performed, and weights were saved from the best of these. The result was one set of weights, which represented the best NN of the 100. Two sets of tests were run. The first was a simple test of how well the NN modeled the system. The second designed controllers using the NN model and the the adaptive critic technique, and tested how well the controller performed.

**System Model Test**

Although simple, the system identification test was extremely difficult for the NN. This is because, while a sigmoid can approximate a step function, it is *not* a step function, and thus has a built-in error. This is the price paid for a differentiable function. The test was based on thirty NN models trained using GA-derived data and thirty NN models trained randomly-generated data. Both sets of models were then tested on both GA-derived and randomly-generated data sets.

**Controller Test**

DHP controllers were created using weights from each technique and tested in each of the demand scenarios.
We made ran ten runs (1,000 controllers total) using a system model NN that had been trained on randomly generated data, and a standard controller-selection heuristic was run to select one test controller (out of 100 per run), and the results compared with a similarly-selected 10-run set of GA-driven controllers.

## 5. Results and Discussion

Results are shown in Tables 1 and 2. The GA-based NN models and the random-based NN models produced accuracies within 2.5% of each other (Table 1), and this difference was evaluated as non-significant, using a two-tailed, two-sample t-Test (n=30).

**Table 1. Impact of System Model Training Data System Model Quality.** Comparison of system models trained on data developed by GA, and models trained on randomly generated data. Significance of the differences was measured with a two-sample, two-tailed t-test (n=30). There was no significant difference

| | **Source of test data** | |
|---|---|---|
| **Source of training data** | **GA** | **Rand** |
| **GA** | 0.422 | 0.344 |
| **Random** | 0.423 | 0.344 |
| **t-Test p-value** | 0.65 (non-sig) | 0.97 (non-sig) |

For the DHP controllers, the scores (in this case the ability to minimize costs, Table 2.) are also close or identical – within 1% when totals for the different tests are averaged. None of the differences were significant.

**Table 2. Impact of System Model Training Data on Total Cost Score.** Comparison of selected controllers using system model NNs trained on data developed by GA, and random processes. Within each category, we compared the scores of GA-derived controllers with that of the random data-derived controllers. Significance of the differences was measured with a two-sample, two-tailed t-test. For example, controllers trained using GA-derived data in a stationary environment had an average arbitrary score of 458. Controllers trained using a random-data system scored 457, and this difference was non-significant at the 0.29 level (n=10)

| | **Controller Performance When Tested On Training Data** | | | **Controller Performance When Tested On Real World Data** | | |
|---|---|---|---|---|---|---|
| | **GA** | **Rand** | **t-Test** | **GA** | **Rand** | **t-Test** |
| **Baseline** | 458 | 457 | 0.29 | 542 | 542 | 0.39 |
| **Stepped Demand** | 422 | 421 | 0.12 | 448 | 446 | 0.18 |
| **Increasing Demand** | 389 | 389 | 0.15 | 502 | 502 | 0.68 |

## 6. Conclusions

The selection of a data source for training an artificial neural net as a system model is a matter of convenience and computational efficiency. Generation of random data is *always* computationally easier, and should be the default technique. However, if the project at hand involves developing a genetic algorithm, the availability of the GA data would argue for using that as the basis for the system model training.

.

## 7. References

[1] Barto, A., Sutton, R., and Anderson, C.: Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems, *IEEE Transactions On Systems, Man, And Cybernetics*, Vol. SMC-13, No. 5, (1983) 834-846.

[2] Bellman, R., and Dreyfus, S.: *Applied Dynamic Programming*. Princeton, Princeton University Press (1962).

[3] Bellman, R., Esogbue, A., and Nabeshima, I.: *Mathematical Aspects of Scheduling and Applications*. New York, Pergamon Press (1982).

[4] Bellman, R., Glickberg, I., and Gross, O.: *Some Aspects of the Mathematical Theory of Control Processes*, R-313. Santa Monica, RAND Corporation (1958).

[5] Bellman, R.: *Dynamic Programming*. Princeton, Princeton University Press (1957).

[6] Caskey, K. R.: *Genetic Algorithms and Neural Networks Applied to Manufacturing Scheduling*. Doctoral Dissertation, Seattle, University of Washington (1993).

[7] Caskey, K., and Storch, R.: Heterogeneous Dispatching Rules in Job and Flow Shops, *Production Planning & Control*, Vol. 7, No. 4, (1996) 351-368.

[8] Hrycej, T.: *Neurocontrol*. New York, John Wiley & Sons (1997).

[9] Ishigami, A., et al.: Structural Control Based on Genetic Algorithm and Neural Network for Electric Power Systems, *Proceedings of the IEEE Applications of Artificial Neural Nets to Power Systems Conference*, (1993). 169-174.

[10] Lin, C., and Jou, C.: Controlling chaos by GA-based reinforcement learning neural network, *IEEE Transactions on Neural Networks*, Vol. 10, No. 4, (1999) 846-859.

[11] Mühlenbein, H., Paaß, G.: From recombination of genes to the estimation of distributions. In Lecture Notes in Computer Science 1411: Parallel Problem Solving From Nature IV, Springer-Verlag, Berlin-Heidelberg, New York (1996) 178-187.

[12] Pelikan M., Goldberg D.E., Lobo, F.: A Survey to Optimization by Building and Using Probabilistic Models. *Computational Optimization and Applications* 21(1), (2002). 1-20.

[13] Pipe, A., Jin, Y., and Winfield, A. A: Hybrid Adaptive Heuristic Critic Architecture for Learning in Large Static Search Spaces. *Proceedings of the IEEE International Symposium on Intelligent Control*. (1994) 237-242.

[14] Prokhorov, D., and Wunsch, D.: Advanced Adaptive Critic Designs, Proceedings of the World Congress on Neural Networks, (1996) 83-87.

[15] Prokhorov, D., Santiago, R., and Wunsch, D.: Adaptive Critic Designs: A Case Study For Neurocontrol, *Neural Networks*, Vol. 8, No. 9, (1995) 1367-1372.

[16] Shervais, S., Shannon, T., and Lendaris, G.: Intelligent Supply Chain Management Using Adaptive Critic Learning, *IEEE Transactions on Systems, Man, and Cybernetics* – Part A, Vol 33, No 2, (2003) 215-244.

[17] Ventura, D., and Martinez, T.: A General Evolutionary/Neural Hybrid Approach To Learning Optimization Problems, *Proceedings of the World Congress on Neural Networks*, (1996) 1091-1095.

[18] Ventura, D., and Martinez, T.: Robust Training Using Training Set Evolution, *Proceedings of the International Congress on Neural Networks*, (1996b) 524-528.

[19] Ventura, D., and Martinez, T.: Using Evolutionary Computation To Generate Training Set Data For Neural Networks, Proceedings of the International Congress on Neural Networks and Genetic Algorithms, (1995) 468-471.

[20] Werbos, P.: Advanced forecasting methods for global crisis warning and models of intelligence, *General Systems Yearbook*, (1977) 25-38.

[21] Werbos, P.: Neurocontrol and related techniques, in Maren, A., Harston, C., and Pap, R., Eds., *Handbook of Neural Computing Applications.* Academic Press, Inc., New York, (1990) 345-380.

[22] Werbos, P.: Neurocontrol and Supervised Learning: An Overview and Evaluation, in White, D., and Sofge, D., Eds., *Handbook of Intelligent Control*. New York, Van Nostrand Rheinhold (1992).

[23] Widrow, B., Gupta, N., and Maitra, S.: Punish/Reward: learning with a critic in adaptive threshold systems, IEEE *Transactions in Systems, Man, and Cybernetics*, Vol. SMC-3, No. 5, September, (1973) 455-465.

[24] Wu, C. : Intelligent use of delayed information in the supply chain by artificial neural network, *Proceedings of the 1999 Conference on Systems, Man, and Cybernetics*, Vol. II, (1999) 66-70.

[25] Wunsch, D., and Prokhorov, D.: *Advanced Adaptive Critic Designs, Computational Intelligence*: A Dynamic System Perspective, IEEE Press, (1995) 98-107.