Proceedings of the International Conference on Neural Networks, pp. 524-528, 1996

Robust Optimization Using Training Set Evolution

Dan Ventura
Tony R. Martinez

Computer Science Department, Brigham Young University, Provo, Utah 84602
e-mail: dan@axon.cs.byu.edu, martinez@cs.byu.edu

ABSTRACT

Training Set Evolution is an eclectic optimization technique that combines evolutionary computation (EC) with neural networks (NN). The synthesis of EC with NN provides both initial unsupervised random exploration of the solution space as well as supervised generalization on those initial solutions. An assimilation of a large amount of data obtained over many simulations provides encouraging empirical evidence for the robustness of Evolutionary Training Sets as an optimization technique for feedback and control problems

1. Introduction

Neural networks (NN) have been successfully applied to a variety of problems [2][5]. Also, work involving a combination of evolutionary computation (EC) and NN is becoming more prevalent [1][4][6][7]. One class of problems to which NN are often applied is that of optimization. The NN is responsible for optimizing a system based upon some criteria. Evolutionary Training Sets is an optimization technique that employs evolutionary computation [3][8] as a preprocessor that creates a training set for the neural network. The only requirements for the optimization technique are access to the system to be optimized and *a priori* knowledge of a fitness function that describes the desired optimization. The synthesis of EC with NN provides both initial unsupervised random exploration of the solution space as well as supervised generalization on those initial solutions.

Evolutionary Training Sets are introduced in [9] and [10]. Two artificial problems have been designed to explore the usefulness of this optimization technique. This paper extends previous work by presenting results obtained from running hundreds of new simulations in order to study the effects of varying periods of evolution and training set size on the effectiveness of optimization. Empirical results provide encouraging evidence for the robustness and general usefulness of Evolutionary Training Sets.

Section two of the paper presents a generalized formal description of the problem to be solved -- optimization of a system (or equivalently, function approximation), either statically or dynamically (control problems). Section three then briefly discusses the combination of a neural network with evolutionary computation as a general approach to solving the problem of section two. Section four presents data (collected over many empirical simulations) in condensed graphical form and discusses its implications for Evolutionary Training Set optimization. Finally, section five presents conclusions and directions for ongoing research.

2. Problem Description

Given a system, Θ, the state of Θ may be described at time t by a vector of status variables, $\boldsymbol{s}^t$. Suppose that control of the system is effected by the setting of variables in a control vector, $\boldsymbol{c}$. That is, given a system Θ at time t described by vector $\boldsymbol{s}^t$, the setting of the values of the vector $\boldsymbol{c}$ will result in a different system Θ' at time $t+\delta$ described by the vector $\boldsymbol{s}^{t+\delta}$. The problem is, given a status vector, $\boldsymbol{s}^t$, what modifications should be made to the control vector $\boldsymbol{c}$ such that $\boldsymbol{s}^{t+\delta}$ describes a better system, if possible, than $\boldsymbol{s}^t$? Obviously, some evaluation or fitness function, f, is necessary in order to determine whether or not one system is better than another.

The operation of Θ may be either continuous or discrete. A neural network is expected to detect the values of $\boldsymbol{s}^t$ and to output values for $\boldsymbol{c}$, the goal being to maximize f for any given instance (status) of Θ. If the problem is an optimization problem, this is a single iteration process; if the problem is an optimization/feedback problem then the process becomes an ongoing series of iterations.

3. Combining Evolutionary Computation with Neural Computation

From the space defined by $\boldsymbol{s}$ that describes Θ we choose a representative set of system states by choosing n initial status vectors. We denote these $\boldsymbol{s}_i^{t=0}$, $0<i\leq n$ and refer to the system state described by $\boldsymbol{s}_i^{t=0}$ as $\Theta_i^{t=0}$, $0<i\leq n$. These choices could of course be biased by any a priori heuristics as to what constitutes a realistic system. In choosing this set of status vectors, $\boldsymbol{s}_i^{t=0}$, $0<i\leq n$, we have chosen the left hand sides of the training instances. We now use evolutionary computation to discover "good" right hand sides, yielding training instances of the form $\boldsymbol{s}_i^{t=0} \rightarrow \boldsymbol{c}_k$.

Assume a fitness function *f* that takes as input a status vector $\boldsymbol{s}$ and returns a real-valued fitness measure. Now for each $\boldsymbol{s}_i^{t=0}$, randomly initialize a population of *m* control vectors, denoted $\boldsymbol{c}_k$, $0<k\leq m$. Evaluate the initial population by simulating the workings of $\Theta_i^{t=0}$ for δ time steps (where δ time steps are sufficient for Θ_i to stabilize) for each $\boldsymbol{c}_k$, and then applying fitness function *f* to $\boldsymbol{s}_i^{t=\delta}$. Next choose parents and use genetic operators to produce *m* offspring. Now evaluate the children and select *m* survivors from amongst the parents and children. Repeat this process until some stopping criterion is met (such as reaching a specified number of generations or finding an individual with a fitness higher than some threshold).

Finally, choose *j* individuals from each of the *n* populations and build a set of *jn* training examples of the form $\boldsymbol{s}_i^{t=0} \rightarrow \boldsymbol{c}_k$. (To avoid ambiguity in the training set we could set *j*=1.)

Since we have only chosen a finite number of seed points from this space, our evolutionary computation has found approximate solutions for only these *n* points in the space and can say nothing about any other points, many of which we are likely to encounter during normal execution of Θ. Therefore it becomes necessary to generalize on this relatively small set of approximate solutions. Using this set of approximate solutions as training examples, an NN model can be trained to develop a general hypothesis over the entire space defined by $\boldsymbol{s}$. (For a more thorough explanation of the algorithm, see [9] or [10]).

The power of this NN/EC hybrid approach is its general applicability to a wide class of problems including function approximation problems, optimization problems, feedback problems, and control problems.

4. Empirical Results

In order to study the probabilistic effects of varying length of evolution and training set size on quality of optimization, simulations using two artificially generated problems were run. The first entails solving a set of mathematical equations (described by a matrix) and the second requires the EC/NN combination to attempt to learn how to hit a target moving in 2-d space. Due to space constraints, neither problem is described in this paper; however, the first is described in [9] and the second in [10]. In general, both simulation processes include the following steps:

1. Generate a problem definition
2. Create a training set using evolutionary computation
3. Train an NN with the training set
4. Create a test set
5. Test the NN on the training set

Since each example in the training set has an associated fitness value (determined during creation of the training set by the evolutionary computation), an overall average fitness of the training set can easily be determined. We term this average fitness of the training set its *quality*. This research attempts to answer the following three questions regarding training set quality:

1) Does the evolutionary computation produce training sets of good quality?
2) Does training set quality correlate with NN generalization?
3) Does number of training examples or quality of training set more greatly affect NN generalization?

Artificial problem generation/simulation programs were used for several reasons. First, they are much easier to work with in terms of analysis, reproduction of results, etc. Second, it is possible to create a test set which can be used to show how well the NN is performing in relation to optimum, and thus to establish (to some extent) the quality of the optimization procedure. Third, fitness of individual training examples and therefore of entire training sets can also be measured against optimum. The training set quality then becomes a natural (though not a strict) upper bound on the NN's generalization accuracy and can therefore be used as a yardstick by which to measure the NN's performance.

All results are averages over ten runs, and all NN simulation was done with the PDP implementation of the back propagation algorithm [5].

4.1. Matrix Problem

This problem involves solving a system of 10 equations with 15 variables for the 5 unknown variables, where the system of equations is represented by a matrix. The evolutionary computation produces a training set of hopefully good solutions for a small number of points (1000 examples) in the equation space; the NN then generalizes on the training set in order to approximate the function described by the matrix. Since the variables are defined on the range [0,100), the problem is difficult because of the huge search spaces (100^{10} for the space to be explored by the EC and 100^5 for the space to be generalized by the NN) involved.

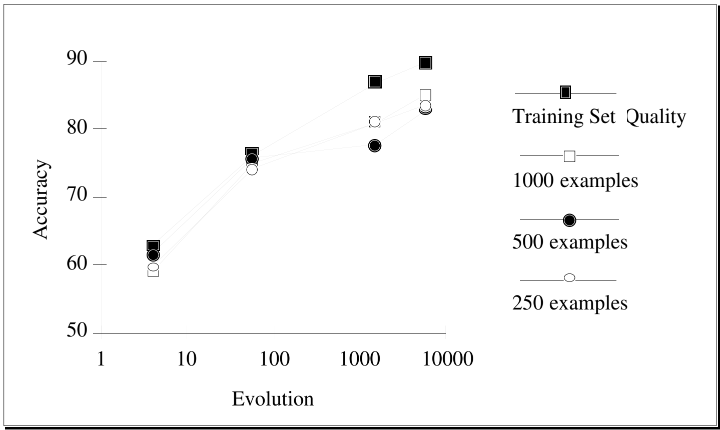

Figure 1. Training set quality and NN generalization accuracy for matrix problem

Figure 1 shows the effects of evolution on training set quality (indicated by the filled squares). Here evolution is indicated on the *x*-axis and is a logarithmic function of both population size and number of generations. The more extended the evolution (that is, the larger the population and/or the more generations of evolution), the higher the training set quality. This is not surprising. The more interesting result is that the higher the training set quality, the better the NN generalization. The hollow squares indicate NN generalization using all 1000 examples in the training set, the solid circles indicate generalization on half of the training set (500 examples), and the hollow circles show generalization on one quarter of the training set (250 examples). Notice that generalization accuracy increases as training set quality increases. In other words, the evolutionary computation is producing a training set that faithfully represents the underlying function to be approximated.

Figure 2 shows that as evolution time increases, the standard deviation in training set quality decreases. So with longer evolution, the probability of finding a good training set increases. Even more encouraging, Figure 2 also indicates that the standard deviation in NN generalization also decreases as training set quality goes up. Therefore, as evolution times are increased, the probability of finding a high quality training set increases; and as the quality of training set increases, the probability of good generalization accuracy increases as well.

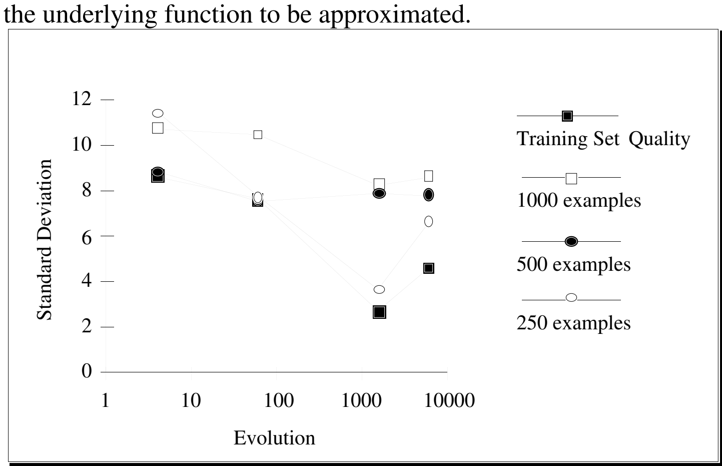

Figure 2. Standard deviation of training set quality and generalization accuracy for matrix problem

4.2. Target Problem

This problem involves the NN trying to learn to hit a target moving in 2-D space with a simple gun, where both bullet and target are subject to the effects of gravity. The evolutionary computation produces a training set of hopefully good bullets for a small number of points (250 examples) in the target space; the NN then generalizes on the training set in order to attempt to learn how to hit any target. The search spaces are much smaller in this problem (35^2 for the EC and 45^2 for the NN); the difficulty this time arises from the fact that the gun is placed in front of the target origin so that some targets cannot be hit at all. This has the effect of introducing noise into the generated training set.

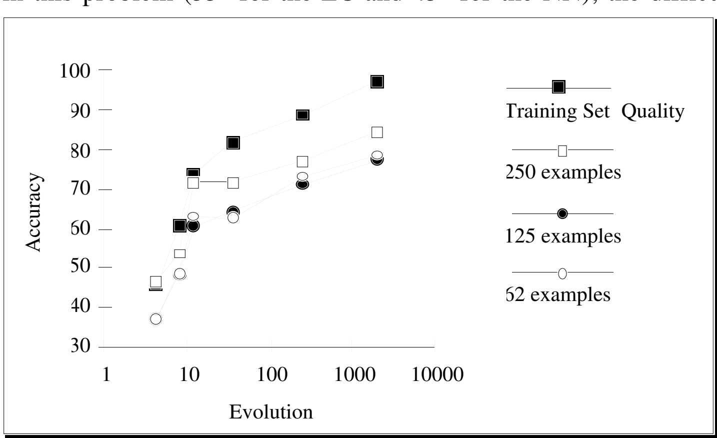

Figure 3. Training set quality and NN generalization accuracy for target problem

Figures 3 and 4 are analogs to figures 1 and 2 for the target problem. Figure 3 shows again both that training set quality increases with time and that NN generalization accuracy increases with training set quality. Further, Figure 4 reiterates the idea that confidence in training set quality increases with time of evolution and also that confidence in generalization accuracy increases with training set quality. Again, it is seen that as evolution time increases, the probability of producing a high quality training set also increases and that as the quality of the training

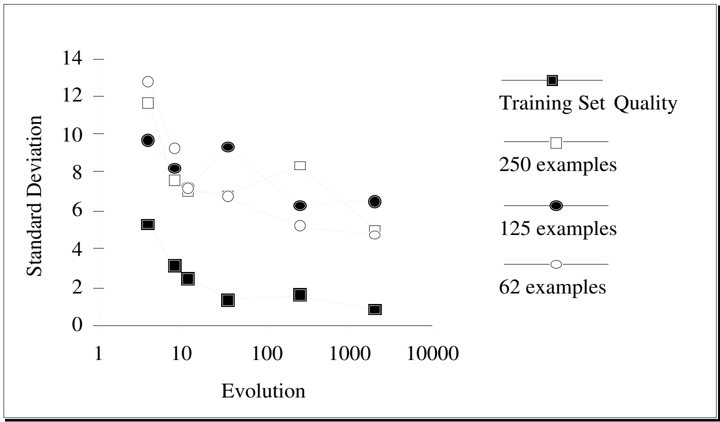

Figure 4. Standard deviation of training set quality and generalization accuracy for target problem

set increases, the probability of good generalization accuracy increases as well.

Another point of interest in both Figures 1 and 3 is the effect that training set quality vs. training set size has on NN generalization. Notice that with a 5% to 10% increase in training set quality the number of training examples required to maintain generalization accuracy is reduced by 50% to 75%. For example, in figure 3, the fourth solid square from the left indicates a training set quality of .820. The resulting NN generalization accuracy on the entire training set (250 examples) indicated by the fourth hollow square is .718. However, when the training set accuracy is increased to .891 (the next solid square to the right), the fifth hollow circle from the left shows that only one quarter of the training set (62 examples) is required to maintain that generalization accuracy (actually it increases slightly to .736).

One difference between Figures 1 and 2 and Figures 3 and 4 is the stratification of training set quality from NN generalization accuracy. This is readily explained by the previously mentioned noise that is inherent in these training sets. Thus, even though the training set quality can be extremely high, NN generalization suffers somewhat because noise exists in the training set. Nevertheless, the principle of longer evolution producing a good training set which results in good NN generalization is still very much in evidence.

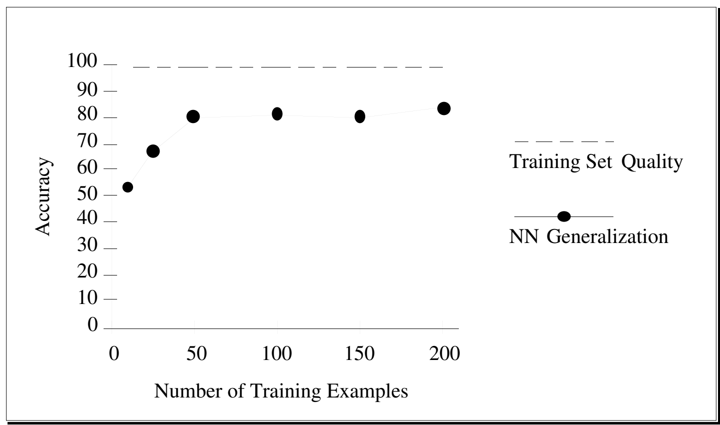

Figure 5. Generalization accuracy vs. number of training examples for target problem

4.3. Training set quality vs. training set size

The effects of quality vs. size were further explored in this final set of simulations. The highest quality (.978 average) training sets generated in the target problem were altered by removing the noisy instances and were then randomly and systematically reduced in size. This was done to investigate the value of individual examples for NN generalization. Figure 5 shows accuracy of generalization vs. number of training examples. With only 50 instances in the training set generalization accuracy is at .80. Since the number of possible targets in this problem is 35^2=1225, this performance is attained after seeing only 4% of the possible targets. Another way of looking at this is to consider the generated training set as a set of exemplars. We have attained 80% of optimum performance while reducing the problem representation by 25 times.

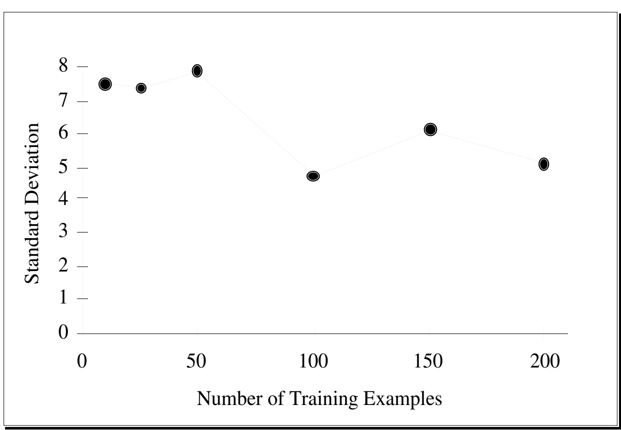

Figure 6. Standard deviation of accuracy vs. number of training examples for target problem

Finally, Figure 6 is included to show the standard deviation in generalization accuracy as opposed to training set size. Even though standard deviation is somewhat higher with a training set size of only 50, it still indicates a confidence in the generalization ability with a training set of that size. Increasing to 100 examples lowers the standard deviation to 5%, indicating that even if the training set is as small as 8% of the total number of possible targets, 95% of the time generalization accuracy will be at least 70% of optimum.

5. Conclusion and Future Work

The results in this paper are an assimilation of data collected over the course of hundreds of EC and NN simulations. They provide empirical evidence that

1) Evolutionary computation produces training sets of good quality.
2) The longer the evolution, the greater the confidence in the training set quality.
3) Training set quality correlates with NN generalization.
4) The higher the training set quality, the greater the confidence in the NN generalization.
5) Training set quality has a greater effect on NN generalization than does training set size.

Current research focuses on developing a theoretical basis for the empirical results discussed in this paper. Also, application of Evolutionary Training Sets to real world problems (such as real-time network control) is necessary to further validate this optimization technique.

References

[1] Caudell, T. P. and Dolan, C. P., “Parametric Connectivity: Training of Constrained Networks using Genetic Algorithms”, *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[2] Falhman, S. E. and Lebiere, F., “The Cascade-Correlation Learning Architecture”, *Advances in Neural Information Processing 2*, D. S. Touretzky (ed.), Morgan Kaufman, 1990.

[3] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing, 1989.

[4] Harp, S. A., Samad, T., and Guha, A., “Designing Application-Specific Neural Networks Using the Genetic Algorithm”, *NIPS-89 Proceedings*, 1990.

[5] McClelland, James L. and Rumelhart, David E., *Explorations in Parallel Distributed Processing*, MIT Press, Cambridge, Massachusetts, 1988.

[6] Montana, D. J. and Davis, L., “Training Feedforward Neural Networks Using Genetic Algorithms”, *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[7] Romaniuk, Steve G., “Evolutionary Growth Perceptrons”, *Genetic Algorithms: 5th International Conference (ICGA-93)*, S. Forrest (ed.), Morgan Kaufman, 1993.

[8] Spears, W. M., Dejong, K. A., Baeck, T., Fogel, D., and de Garis, H., “An Overview of Evolutionary Computation”, *European Conference on Machine Learning (ECML-93)*, 1993.

[9] Ventura, Dan, Andersen, Tim, and Martinez, Tony R., “Using Evolutionary Computation to Generate Training Set Data for Neural Networks”, *Proceedings of the International Conference on Neural Networks and Genetic Algorithms*, pp. 468-471, 1995.

[10] Ventura, Dan, and Martinez, Tony R., “A General Evolutionary/Neural Hybrid Approach to Learning Optimization Problems”, submitted to the *World Congress on Neural Networks*, 1996.