An Evolutionary Algorithm for Function Inversion and Boundary Marking

Russell D. Reed, Robert J. Marks II
Dept. of Electrical Engineering, FT-10
University of Washington, Seattle, WA 98195, USA

ABSTRACT

We present an evolutionary algorithm for distributing points evenly on a surface $f(\mathbf{x}) = c$. Applications include function inversion (where system inputs are sought which produce a desired output) and boundary detection for e.g. control systems or power-system security assessment, in which boundaries of a safe operating region are to be avoided.

1 Introduction

In many control applications it is necessary to ensure that the system stays within a safe operating region. If the system approaches the boundary we would like a warning to be generated and would like to know how to adjust the operating point to avoid the boundary. If the system goes outside the boundary, we would like to know the quickest way back in. In other applications, it is useful to be able to invert a function $f(\mathbf{x}) = y$ to find the values $\mathbf{x}$ which produce a desired output y.

Function inversion may be useful in neural network training since training points near the classification boundary may provide more useful information about how the network needs to be adjusted than points distant from the boundary. A gradient based neural network inversion algorithm is described and used for query-based learning in [5, 6]. Use of a genetic algorithm for inversion of a neural network is described in [1].

Closed-form solutions describing a boundary surface can sometimes be found when $f(\mathbf{x})$ is relatively simple, but may be intractable when $f(\mathbf{x})$ is a complex nonlinear function. In the following, we describe an evolutionary algorithm for distributing a set of points approximately evenly on a p-dimensional surface defined $f(\mathbf{x}) = c$.

2 The Algorithm in General Terms

The goal of the algorithm is to distribute a set of points evenly on a manifold defined by $f(\mathbf{x}) = c$, where $f(\mathbf{x})$ is some generating function, $\mathbf{x}$ is a vector in R^p, and c is a constant (or slowly changing). The level sets $f(\mathbf{x}) = c$ can be thought of as contours of the function analogous to contour lines on a geological map. Depending on $f(\mathbf{x})$, the surface may be nonlinear and disconnected. Differentiability of $f(\mathbf{x})$ isn't required, but gradient information can be used if available.

An evolutionary procedure is used; a large number of points are generated at random positions, the worst are eliminated and replaced by variations of the best. After many iterations, the population evolves to a state that satisfies the governing constraints.

'Birth' and 'death' of the points is controlled by two basic goals: (1) the points should lie on the surface, and (2) the points should be evenly distributed over the surface. The first goal is achieved by periodically eliminating points with the worst functional errors $e(\mathbf{x}) = |f(\mathbf{x}) - c|$. The second goal is achieved by generating their replacements from perturbations of uncrowded points (which are well separated from their neighbors) and by repulsion between nearest neighbors. Since the uncrowded points survived elimination their functional errors are likely to be smaller so the error tends to decrease as the dispersion increases.

The algorithm is similar to the evolutionary strategy of [7] but the two subgoals are satisfied by two different mechanisms rather than embedded in a single fitness function which might have undesired local minima. Unlike the genetic algorithm, reproduction is 'asexual' —there is no mating or string crossover.

In very loose terms, an analogy could be made to certain species of marine life (e.g., barnacles, crabs, snails) adapted to conditions in the shoreline tidal zone. Individuals that stray too far from the water line (have high $|f(\mathbf{x}) - c|$ errors) suffer and do not reproduce. Within the tidal zone, food and shelter are limited so individuals finding relatively uncrowded areas thrive.

Table 1: Variable definitions

Variable	Meaning
p	dimension of the space
$\mathbf{x}_i$	position of point i, $\mathbf{x} \in R^p$
$f(\mathbf{x})$	the generating function
c	the contour defining the surface, $f(\mathbf{x}) = c$
N	population size
M	number of points replaced per generation
σ	noise level used to generate new points from old points
m	number of nearest neighbors used to measure crowding
s	stepsize for repulsion from neighbors
d_i^m	average distance from point i to its m nearest neighbors

3 The Algorithm

The process starts with N points randomly distributed over the space of interest. In each generation, M points with the worst functional error $|f(\mathbf{x})-c|$ are deleted and replaced by perturbations of the least crowded remaining points. Details are outlined below.
Generate N points randomly distributed (e.g., uniformly) over the space of interest. For each generation:

1. sort the points by their $|f(\mathbf{x}_i) - c|$ errors
2. delete the M points with the worst $|f(\mathbf{x}_i) - c|$ errors
3. generate a replacement for each deleted point:
   (a) sort the remaining points in order of d_i^m, their average distance to their nearest m neighbors,
   (b) select a parent k from the least crowded points. Random selection from the first $N/5$ points in the d_i sort order is typical.
   (c) generate the new point $\mathbf{x}_{new} = \mathbf{x}_k + \mathbf{n}$ where $\mathbf{n} \sim N(0, \sigma\mathbf{I})$, for example.
4. (optional) Repel crowded points. For each point x_i among the most crowded $N/5$ points:
   (a) Calculate the vector $\mathbf{v}$ away from its nearest neighbor $\mathbf{x}_k$
   $$\mathbf{v} = \mathbf{x}_i - \mathbf{x}_k$$
   and the gradient vector
   $$\mathbf{g} = \frac{\partial f}{\partial \mathbf{x}}$$
   $$\hat{\mathbf{n}} = \mathbf{g}/||\mathbf{g}||.$$
   ($\hat{\mathbf{n}}$ is a unit vector normal to the surface.)
   (b) Project $\mathbf{v}$ onto the subspace orthogonal to $\mathbf{g}$ (i.e., remove the component of $\mathbf{v}$ parallel to $\mathbf{g}$ so that the result is parallel to the contour and the new point will have the same approximate $f(\mathbf{x})$ value)
   $$\mathbf{u} = \mathbf{v} - \mathbf{v}^T\hat{\mathbf{n}}$$
   $$\Delta\mathbf{x} = s \cdot \mathbf{u}/||\mathbf{u}||$$
   where s is small constant step size, e.g. $s = 0.01$.
   (c) Replace $\mathbf{x}_k$ with $\mathbf{x}_k + \Delta\mathbf{x}$. Since $\Delta\mathbf{x}$ is a vector parallel to the surface, $f(\mathbf{x})$ should not change much when s is small.

Example
Fig. 1 shows a simple 2–dimensional example to illustrate how the algorithm works. The function is the sum of two tanh functions:

$$f(x,y) = [\tanh(2 - 4x + 2y) + \tanh(1 + 1x - 2y)]/2$$

The target contour is $c = 0.5$. Other parameters are $N = 50$, $M = 3$, $\sigma = 0.05$, $m = 2$, $s = 0.01$. Fig. 1a shows the distribution after a few generations, a-d show snapshots at selected iterations. As the system evolves, points with the worst errors are eliminated. Replacements are generated from perturbations of points that survived (and so have smaller functional errors $|f(\mathbf{x}) - c|$) so the average error in the next generation tends to decrease if σ is not too large.

Fig. 2 shows another example with a multi–modal function ($c = 0.9$, $N = 100$, $\sigma = 0.05$, $M = 5$, $m = 2$, $s = 0.01$).

4 Remarks

Selection of σ
Initially, a reasonably large σ is desirable to explore the space. In later stages though, the worst $|f(\mathbf{x}) - c|$ error will be small and it may be desirable to decrease σ in order to increase the survival rate of offspring. That is, the average distance of a new point from its parent is σ. Using a linear approximation, $\Delta f = f_{new} - f(x_k) \approx ||\mathbf{g}(\mathbf{x}_k)||\sigma$ where $\mathbf{g}(x_k) = \partial f/\partial\mathbf{x}$ evaluated at $\mathbf{x}_k$. If the surface is steep at $\mathbf{x}_k$ and σ is large then Δf will be large and the error is likely to be high leading to deletion of the point in the next generation. On the other hand, if σ is small then $\mathbf{x}_{new}$ will likely be close to its parent; the points may then gather in small clumps and not be well dispersed over the surface.

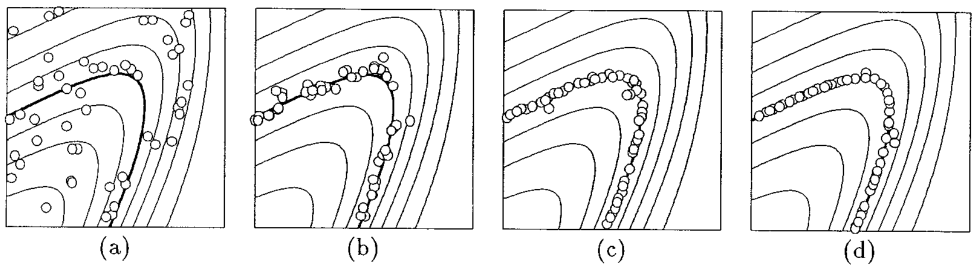

Figure 1: Evolution of the population for a simple 2D function.

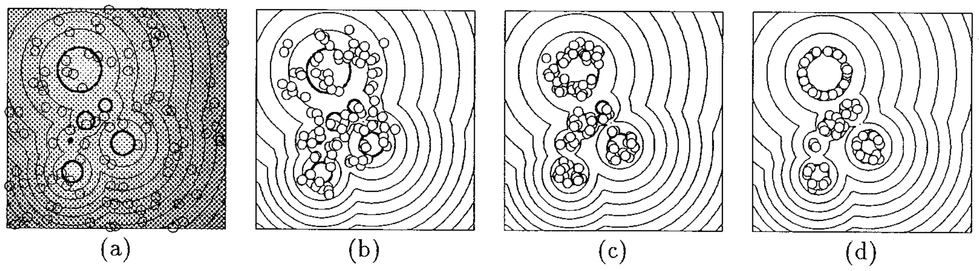

Figure 2: Evolution of the population for a multi–modal 2D function.

Dispersion

Repulsion between nearest neighbors helps to avoid clumping. In repelling crowded points, the recipe above uses gradient information to project $\mathbf{v} = \mathbf{x}_i - \mathbf{x}_k$ onto the surface tangent plane. Gradient information isn't necessary, however. The simpler form, $\Delta\mathbf{x} = s\,\mathbf{v}/||\mathbf{v}||$, where points simply move directly away from their nearest neighbor, gives similar results in the final stages when most of the points are already nearly on the surface and so $\mathbf{v}$ is nearly parallel to it. In areas where the surface has high curvature and points are widely separated, however, this may cause new points to fall outside the error tolerance band.

Number of Neighbors

The m nearest neighbors, rather than just 1, are used to measure crowding in order to smooth the distribution. This seems to be a relatively unimportant factor since $m = 1$ works reasonably well in low dimensions. It may be more important in higher dimensions.

Relation to Genetic Algorithm

This algorithm is slightly different from other evolutionary algorithms in that different fitness functions control which points reproduce or die. In basic genetic algorithms, for example, fitness is determined by a single fitness function $F(\mathbf{x})$; points with low F values are deleted and replaced by combinations of points with high F values. Here, points with poor $|f(\mathbf{x})-c|$ values are deleted and replaced by variations of points with high d values. The intention is to minimize the average error $|f(\mathbf{x}) - c|$ and maximize the average distance between neighboring points without having to define a single fitness function which may have undesired local minima.

More importantly, the goal of the algorithm is to optimize a function of the *entire population* rather than a function of a single point. In the basic genetic algorithm, each unit represents an alternative solution to the problem and the end product is a single unit representing the best solution. Since the remaining units tend to be similar to the winner, the population forms a cluster. Variations of GA have been proposed that attempt to preserve genetic diversity by various methods such as niching, sharing, or maintaining separate subpopulations with limited mixing ([2] summarizes a few methods), but they fight this convergence–to–a–point characteristic. It might be possible to use GA or some other optimization method by treating entire populations as points in a larger space, but this could be extremely inefficient if not handled carefully.

Variations

There are many possible variations of the basic algorithm. One possibility is to minimize $|f(\mathbf{x}) - c|$ for each point using gradient information and maximize dispersion by repulsion from nearest neighbors along the contour. A problem is that this may settle to a static distribution in which the final state is completely determined by the initial distribution. In this algorithm, randomness (due to random selection of ~~the parent and the~~ noisy perturbation) helps the system explore the space more effectively. Another problem is that if $|f(\mathbf{x}) - c|$ is minimized by gradient following methods, then every point will eventually arrive at an attractor; since some attractors may have much larger basins of attractions than others, the points are not likely to be evenly distributed on the surface. Repulsion from neighbors may help disperse the points but won't allow movement between two disconnected pieces of the surface. (I.e., it would not even out differences in population density between isolated 'islands' such as in Fig. 2.)

Other possible variations include:

- σ adaptation, possibly using different values for directions parallel and orthogonal to the local gradient depending on the local steepness of $f(\mathbf{x})$
- population size adaptation
- adaptation of the crowding factor, possibly allowing more crowding where the surface curvature is high (to better delineate high curvature regions)

These may have significant effects on convergence time and the quality of the solution, but the basic behavior remains the same.

One form of adaptation that is in keeping with the evolutionary approach of this algorithm is to have offspring inherit their parameters from their parent with perturbations (i.e., mutations) to explore the parameter space.

It is interesting to note that the condensed nearest neighbor classification algorithm [4, 3] also tends to find points near a boundary. Given a set of points in two classes (inside or outside the surface in this case), it keeps only those which are misclassified by their nearest neighbor —a process which tends to favor points near the boundary. This doesn't necessarily disperse the points evenly over the surface, however, and a very large initial set of points might be needed to get a good definition of the border.

Dimensionality

Since the procedure evaluates $f(\mathbf{x})$ many times (once for every new point), it may not be efficient when this is a costly operation. This is characteristic of many evolutionary algorithms, however. In such cases, a neural network or other approximation system might be useful.

It should be noted that the number of points needed to outline a boundary may be very large in even moderately high dimensional spaces. To some extent, this is a 'problem of the problem' given the goal of placing a set of points on the surface. For safe–operating–region problems (where the ultimate goal is to warn of boundary crossings), a two–level procedure is envisioned: points in the initial set are used as markers telling which portion of the boundary is closest and a more precise second–level search (with this technique or some other) is done to get a more precise picture of the boundary in this region. It isn't necessary that the initial points outline the boundary with high precision. If the surface is not too complex, then a tractable number of points may suffice regardless of the fact that they may be widely separated.

References

[1] R. C. Eberhart. The role of genetic algorithms in neural network query-based learning and explanation facilities. In *COGANN-92. International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 169–183, IEEE Computer Society Press, Los Alamitos, CA, 1992.

[2] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, & Machine Learning.* Addision–Wesley, Reading, MA, 1989.

[3] D.J. Hand. and B.G. Batchelor. An edited condensed nearest neighbor rule. *Information Sciences*, 14:171–180, 1978.

[4] P.E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, IT–14(3):515–516, May 1968.

[5] J.N. Hwang, J.J. Choi, S. Oh, and R.J. Marks II. Classification boundaries and gradients of trained multilayer perceptrons. In *IEEE International Symposium on Circuits and Systems*, pages 3256–3259, IEEE, 1990.

[6] J.N. Hwang, J.J. Choi, S. Oh, and R.J. Marks II. Query-based learning applied to partially trained multilayer perceptrons. *IEEE Transactions on Neural Networks*, 2(1):131–136, 1991.

[7] I. Rechenberg. Evolution strategy. In J.M. Zurada, R.J. Marks II, and C.J. Robinson, editors, *Computational Intelligence Imitating Life*, pages 147–159, IEEE Press, 1994.