Evolving Networks:
Using the Genetic Algorithm with Connectionist Learning

Richard K. Belew
John McInerney
Nicol N. Schraudolph

Cognitive Computer Science Research Group
Computer Science & Engr. Dept. (C-014)
Univ. California at San Diego
La Jolla, CA 92093
rik@cs.ucsd.edu

CSE Technical Report #CS90-174

June, 1990

Abstract

It is appealing to consider hybrids of neural-network learning algorithms with evolutionary search procedures, simply because Nature has so successfully done so. In fact, computational models of learning and evolution offer theoretical biology new tools for addressing questions about Nature that have dogged that field since Darwin [Belew, 1990]. The concern of this paper, however, is strictly artificial: Can hybrids of connectionist learning algorithms and genetic algorithms produce more efficient and effective algorithms than either technique applied in isolation? The paper begins with a survey of recent work (by us and others) that combines Holland's Genetic Algorithm (GA) with connectionist techniques and delineates some of the basic design problems these hybrids share. This analysis suggests the dangers of overly literal representations of the network on the genome (e.g., encoding each weight explicitly). A preliminary set of experiments that use the GA to find unusual but successful values for BP parameters (learning rate, momentum) are also reported. The focus of the report is a series of experiments that use the GA to explore the space of initial weight values, from which two different gradient techniques (conjugate gradient and back propagation) are then allowed to optimize. We find that use of the GA provides much greater confidence in the face of the stochastic variation that can plague gradient techniques, and can also allow training times to be reduced by as much as two orders of magnitude. Computational trade-offs between BP and the GA are considered, including discussion of a software facility that exploits the parallelism inherent in GA/BP hybrids. This evidence leads us to conclude that the GA's *global sampling* characteristics compliment connectionist *local search* techniques well, leading to efficient and reliable hybrids.

Contents

List of Figures

1 Introduction

It is extremely appealing to consider hybrids of neural-network-based learning algorithms with evolutionary search procedures, simply because Nature has so successfully done so. In fact, new computational models of learning and evolution offer theoretical biology new tools for addressing questions about Nature that have dogged that field since Darwin [Belew, 1990, Kauffman and Smith, 1986]. However, these same models have proven interesting enough to computer scientists that they can also be treated as artificial *algorithms*, divorced from the natural phenomena from which the models originally sprung. Considered separately, both connectionist networks and "evolutionary algorithms" have recently drawn a great deal of attention as new forms of adaptive algorithhm. On occasion, the two techniques have been compared (e.g., [Brady, 1985]). The concern of the current paper is the composition of these two types of algorithms: Can hybrids of connectionist learning algorithms and genetic algorithms produce more efficient and effective algorithms than either technique applied in isolation? This proves to be a very broad question, and the present paper attempts to provide only a survey of results to date. Based on these experiences, we also identify several key areas for further investigation.

Section 2 begins with a brief description of Holland's Genetic Algorithm (GA). While using the GA to guide connectionist learning systems through specification of the networks' *structural* characteristics is perhaps the most natural, there are other hybrids of the two techniques that also seem promising. For example, Section 4 describes experiments that use the GA to find good values for two critical parameters of the BP learning algorithm, learning rate (η) and momentum (α).

Section 5 considers potential hybrids of GA and connectionist algorithms from the perspective of the state spaces they search and their respective methods. In brief, the GA proceeds by *globally sampling* over the space of alternative solutions, while gradient techniques — including BP but also methods like conjugate gradient — proceed by *locally searching* the immediate neighborhood of a current solution. This suggests that using the GA to provide good "seeds" from which BP then continues to search will be effective. We describe several experiments in which the GA is used to select initial values for the vector of weights used by BP and also by conjugate gradient.

Section 6 briefly sketches some of the computational complexity issues arising from GA/BP hybrids. A software facility that exploits the natural parallelism when the GA is used to control multiple instantiations of BP simulation is discussed, and some features of the time and space complexity of the hybrid systems are considered.

2 Genetic algoritms

The GA has been investigated by John Holland [Holland, 1975] and students of his for almost twenty years now, with a marked increase in interest within the last few years [Grefenstette, 1985,Grefenstette, 1987,Schaffer, 1989]. The interested reader is advised to begin a more thorough introduction to these algorithms with the excellent new text by Goldberg [Goldberg, 1989].

Attempts to simulate evolutionary search date back as far as the first attempts to simulate neural networks [Fogel et al., 1966]. The basic construction is to consider a population of individuals that each represent a potential solution to a problem. The relative success of each individual on this problem is considered its **fitness**, and used to selectively reproduce the most fit individuals to produce similar but not identical offspring for the next generation. By iterating this process, the population efficiently samples the space of potential individuals and eventually converges on the most fit.

More specifically, consider a population of N individuals x_i, each represented by a **chromosonal** string of L **allele** values. An initial population is constructed at random; call this **generation** g_0. Each individual is evaluated by some arbitrary **environment** function that returns the fitnesses $\mu(x_i) \in \Re$ of each individual in g_0. The evolutionary algorithm then performs two operations. First, its **selection** algorithm uses the population's N fitness measures to determine how many offspring each member of g_0 contributes to g_1. Second, some set of **genetic operators** are applied to these offspring to make them different from their parents. The resulting population is now g_1, these individuals are again evaluated, and the cycle repeats itself. The iteration is terminated by some measure suggesting that the population has converged.

A critical distinction among simulated evolutionary algorithms is with respect to their genetic operators. Often the only genetic operator used is

mutation: some number of alleles in the parent are arbitrarily changed in the child. This amounts to a *random* search around the most successful individuals of the previous generation, and is therefore not very powerful. The use of a simple mutation operator, coupled with the exponential amplification of good solutions afforded by selective reproduction, produce a powerful adaptive system on their own and some find this sufficient [Fogel et al., 1966,Rizki and Conrad, 1986].

The central feature of Holland's GA is its use of an additional **cross-over** operator modeled on the biological operation of genetic recombination: during sexual reproduction segments from each of the parents' chromosomes are combined to form the offspring's. One standard version[1] of the cross-over operation picks two points $1 \leq m, n \leq L$ at random and builds the offspring's bit string by taking all bits between m and n from one parent and the remaining bits from the other parent. For example, if $L = 10, m = 2, n = 6$:

```
Parent(1): 1111111111   Offspring(1): 1100001111
Parent(2): 0000000000   Offspring(2): 0011110000
```

The appeal of the GA is due both to empirical studies that show the cross-over operator works extremely well on real, hard problems, and also to the "schemata" analysis Holland has provided to show why this is the case. Briefly, Holland's Schemata Theorem [Holland, 1975, Thm. 6.2.3] suggests that the initially random sampling of early generations is concentrated by the GA's search towards those areas of the search space demonstrating better-than-average performance.

At the same time, the crossover operator imposes severe constraints on the genomic representation, as the experiments with the representation of connectionist networks here will demonstrate. Conversely, modern genetics continues to uncover biological mechanisms that are potentially even more powerful operators than crossover [Huynen and Hogweg, 1989]. This paper, however, will restrict itself to the GA on the grounds that it currently provides the best balance between empirically demonstrated adaptive power and theoretical understanding.

One key property of the GA is that it works on a population of (binary)

[1] Because it has proven such an important component of the GA, many other variations of cross-over are under active investigation. For example, one, two or more cross-over points can be selected; these points can be selected non-uniformly over the string, etc.

bit strings with absolutely no knowledge of the semantics associated with these bits. Its only contact with the environment is the global fitness measure associated with the entire string. This is considered an advantage of the algorithm because it ensures that the GA's success is not related to the semantics of any particular problem. This is not to say that the GA works on all problems equally well, only that these differences can be attributed to the underlying search spaces rather than the semantics of the problem domain [Bethke, 1981].

Having committed ourselves then to the GA and its crossover operator, it is worth noting some of the central "pearls of GA wisdom" that are most salient to the problem of encoding networks onto GA strings. First, our encoding must respect the *schemata*: The representation must allow discovery of small "building blocks" from which larger, complete solutions can be formed. Ideally, this means that we should be able to prove that the Schema Theorem holds with respect to our representation of a network. Second, the well known phenomena of *linkage bias* insists that we do our best to reflect functional interactions with proximity on the string. For example, a great deal of connectionist work has highlighted the role of individual hidden units; localizing the representation of these hidden units on the GA string therefore seems one reasonable strategy. [Merrill and Port, 1988,Montana and Davis, 1989]. Finally, we must worry about the *closure* properties of the GA operators on the network descriptions. It is not strictly necessary that these operators produce valid network descriptions. But unless invalid descriptions are the exception and not the rule, the GA will not get the information about regularities among valid solutions in each new generation it needs to function properly.

3 Mapping networks onto GA strings

Within these basic guidelines, the ways of representing a weighted graph with a string of bits are limited only by the imagination. One useful dimension along which these alternatives can be organized is what Todd has called "developmental specification": i.e., how complete and literal a representation of the network is encoded on the GA string [Todd, 1988]. At one extreme, it is possible to encode each of the network's weights, in full precision, and then use the GA to solve this as a standard multi-parameter function optimization problem; in this case, there is no role for connectionist

learning. At the other extreme, we could emulate biology and use the genetic description as input to a complex *developmental intepretation process* that then constructs the network/phenotype. Between these two extremes are a wide range of mappings in which the GA is used to constrain but not completely determine a network's structure, with connectionist learning processes subsequently embellishing this partial solution.

Surely the most straightforward representation of a connectionist network in a GA string is formed simply by concatenating all of the network's weights.[2] This approach leads to two types of design decision. First, how are each of the real number weights to be represented? Second, in what order are the weights to be concatenated?

3.1 Encoding real numbers

An immediate design issue facing any connectionist/GA hybrid is how the connectionist weights are to be represented on the GA string. Appropriate representation for real values on the basically discrete, binary GA chromosome is a matter of considerable debate within the GA community. Perhaps for this reason, several of the earlier attempts to use the GA with connectionist networks have left real numbers as discrete elements in their representation, thus avoiding this encoding issue [Montana and Davis, 1989, Whitley and Hanson, 1989]. The GA is then allowed to search for good combinations of weights, but is not used for finding the value of any one weight. But this *a priori* division of effort is something we hope to avoid; the many successes with which the GA has been used to discover real-valued quantities suggests that it is also unnecessary [DeJong, 1980].

Figure 1hows the basic features of weight encoding. For each weight, assume first that the real number to be found exists somewhere in the bounded region $[m, M]$. Assume also that B bits have been allocated to represent each weight; these are sufficient to divide the bounded region into 2^B intervals. The B bit index is then Gray-coded to minimize the Hamming distance between indices close in value [Caruana and Schaffer, 1988]. Within the specified interval, then, a real number is selected from a random variable uniformly distributed over that interval. Note that this stochastic element

[2] Some tentative results suggest that with this encoding the GA can find weights more quickly than back propagation, but only on fairly deep networks (i.e., with many hidden layers) [Offutt, 1989].

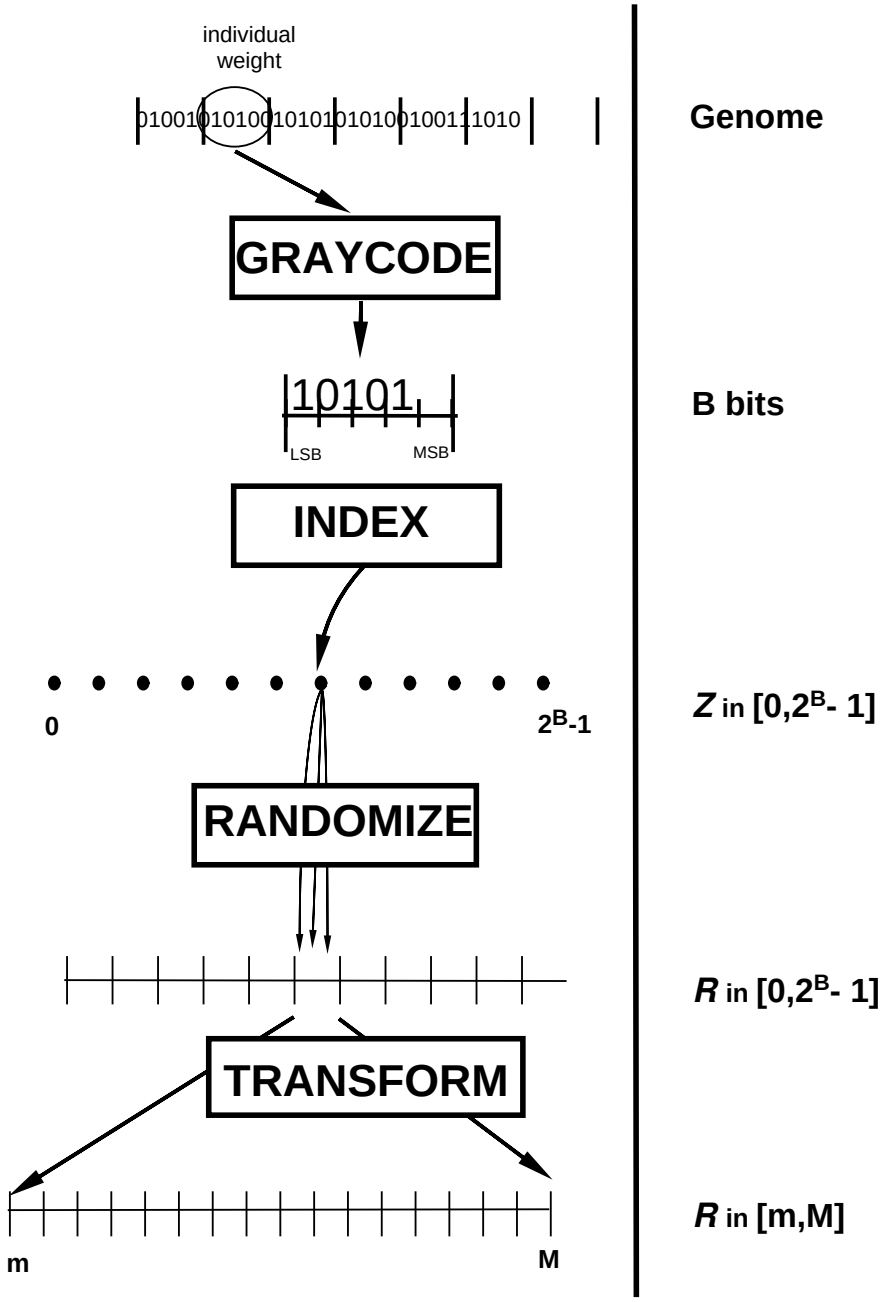


Figure 1: Encoding weights

will often introduce great variability in the resulting networks' performance, as connectionist networks have been shown to be extremely sensitive to small changes in weights [Kolen and Pollack, 1990]. More constructively, it would be desirable if this encoding (the range encoded, the number of bits used) were varied as a consequence of the variability experienced across the population. This kind of *dynamic parameter encoding* for the GA is being explored separately [Schraudolph and Belew, 1990].

3.2 Crossover with distributed representations

Perhaps the most important feature of representation for the GA is proximity. Because two alleles are much more likely to become separated by a crossover operation if they are far apart on the string than if they are close together, it becomes less and less likely that the GA will be able to discover and exploit nonlinear interactions between any two alleles as they are put farther apart on the string. In our case, this lesson suggests that the best representation will have dependent weights close together on the string just if these weights are functionally dependent on one another.

Consider a standard three-layer, feed-forward network. At least in these networks, the obvious functional units correspond to units in the hidden layer. This suggests that all weights associated with one hidden unit should be placed together on the string. Merrill has performed experiments that substantiate this [Merrill and Port, 1988]. Such functional units can be made even more cohesive by introducing "punctuated" crossover operations, which have higher probability of breaking the chromosome at certain punctuated points in the string (e.g., between one hidden unit's weight and another's) [Schaffer and Morishima, 1985].

One important property of the solutions learned by networks, however, is that they are generally far from unique. In the context of the GA, this means that crossover among two relatively good parents who have discovered different solutions can lead to abysmal offspring. Consider again the example of a simple three-layer, feed-forward back propagation (BP) network, and consider the solutions it might discover to the "encoder" problem.[3] A typical solution, reported in the PDP volumes [Rumelhart et al., 1986, p. 337], is

[3] The encoder problem involves mapping N orthogonal patterns through a hidden layer of $log_2 N$ hidden units onto a set of N orthogonal output patterns [Rumelhart et al., 1986, p. 335].

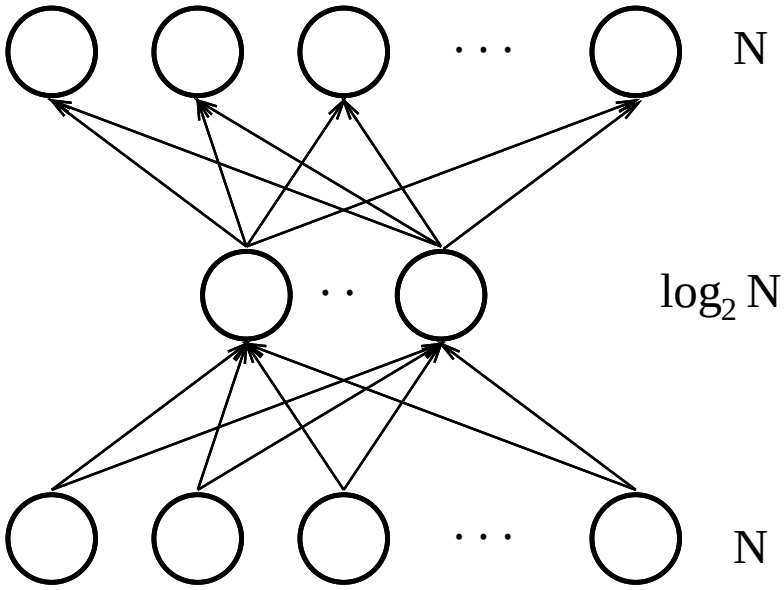


Figure 2: Solution to the encoding problem

shown in Figure 2. Note that while this network discovered the same binary encoding scheme a computer scientist might suggest, it also made use of *intermediate* activation values. In general, we can expect such individual variation among the solutions found by connectionist networks, and so in their corresponding genetic descriptions.

Less subtle but just as problematic variations arise because fully isomorphic solutions can be obtained simply by permuting the hidden units. That is, two networks can be identical up to the arbitrarily assigned *indices* of their hidden units. But (at least in the representations considered heretofore) the location of a hidden unit weight on the GA chromosome depends entirely on its (arbitrarily assigned) index!

The invariance of BP networks under permutation of the hidden units is such a devastating and basic obstacle to the natural mapping of networks onto a GA string that we might consider ways of normalizing network solutions prior to crossover. It seems, for example, that at least in the case of BP networks with a single hidden layer, the differential weighting of the hidden units to the "anchored" (i.e., constant and nonarbitrary) input and output layers might be used to recognize similarly functioning hidden units.

Even establishing a correspondence among hidden units of two three-layer networks that have been trained to solve the same problem appears to be computationally intractable, even when we assume that the only difference between the two networks' solutions is a permutation of hidden units. In realistic networks many other different solutions to the same problem can be constructed, for example by reversing the sign of all weights, or taking other "semi-linear combinations"[4] of the weights. We therefore conclude that attempting to normalize networks before combining them is not feasible.

Thus, the specifics of a network's architecture are *underdetermined* by the problem it is trying to solve. Consequently the genetic representations of these varying architectures cannot be expected to share the similarities (schemata) that the GA needs in order to be effective. If very small populations are used with the GA, there is not "room" for multiple alternatives to develop. In this case, whichever solution is discovered first comes to dominate the population and resist alternatives. This approach has been used successfully by Whitney [Whitley and Hanson, 1989]. Alternatively, the correspondence between genotype and phenotype can be made less direct; the first step in this direction is discussed in the next section.

3.3 Wiring diagrams

As we move away from full specification of all network weights on the genetic string, the goal will be to use the GA to specify some *constraints* on a network architecture. Within these constraints, the connectionist learning procedure then does its best to optimize the objective function via weight modification.

One of the most straightforward architectural descriptions for a network is a binary "wiring diagram." The links of a three-layer feedforward network with I input units, H hidden units and O output units are encoded as a binary string of length $H * (I + 1 + O) + O$, with all links (including the bias) from and to one hidden unit falling contiguously. This wiring specification is given to a simulator that uses BP to do its best to learn the specified task from a series of training examples, and the network's final mean squared error (MSE) becomes the fitness associated with that individual. An entire population of such individuals form a generation. The

[4] We speak a bit loosely here. Because BP networks depend on nonlinear "squashing" functions, simple linear combinations are not quite adequate.

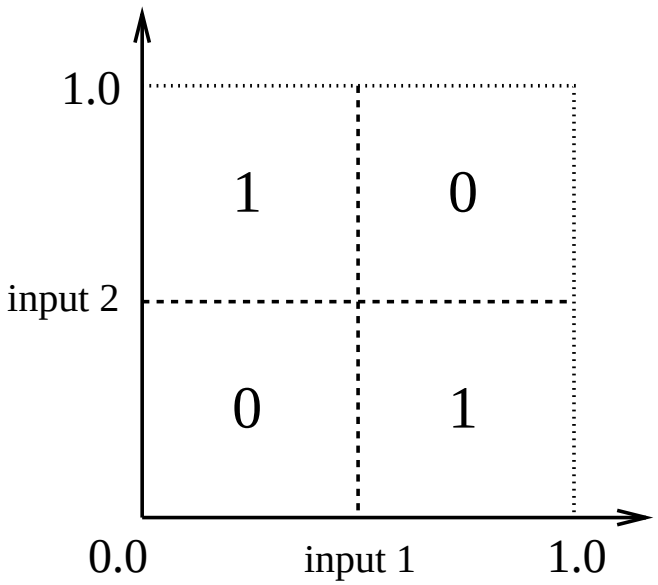


Figure 3: Four-quadrant problem

GA is used to replicate and alter the binary network descriptions to form a new population, and the cycle is then repeated.

Miller et al. report on experiments using this sort of wiring diagram [Miller et al., 1989]. Their most striking result was with the Four-quadrant problem, a generalization of the XOR problem to a two-dimensional real interval; see Figure 3. This problem is interesting because it admits at least two types of solution, as shown in Figure 4. The "fat" solution uses only a single layer of hidden units, with the number of hidden units required growing as the desired precision increases. The "skinny" solution makes use of two layers of exactly two hidden units each. Each input unit is connected to only one of the hidden units in the first layer; this layer simply changes the real inputs to binary values. The rest of the network can then solve the problem as a standard XOR network. Since tradeoffs between wide and deep networks are an important and open issue in connectionist learning systems, this problem is particularly valuable because it provides some experience with using the GA to design networks with more than one hidden layer.

Using a network architecture description that allowed either of these solutions to form, the GA in fact consistently found intermediate solutions

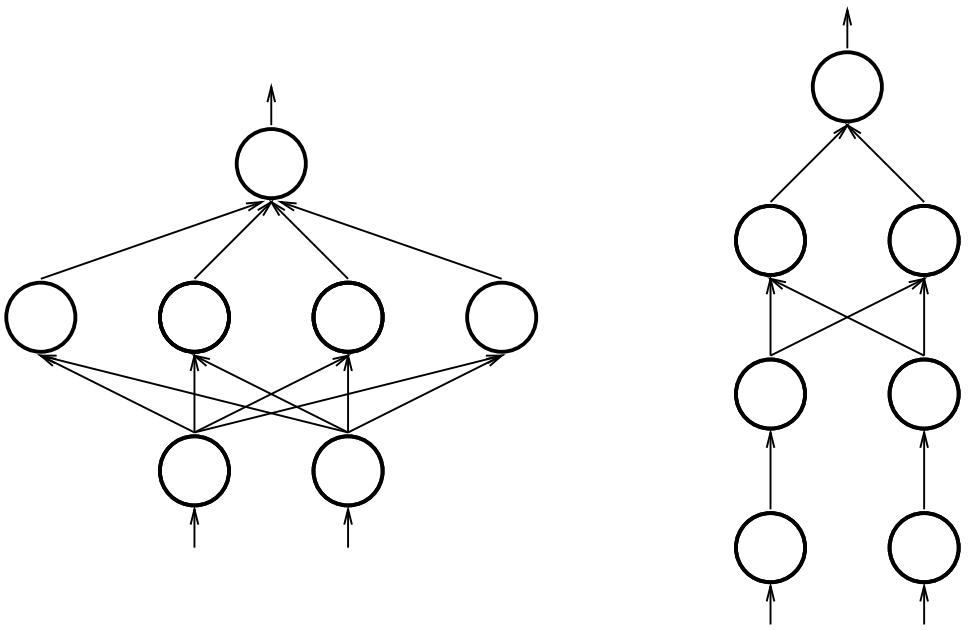


Figure 4: Two solutions to the Four-quadrant problem

like that shown in Figure 5. Note that while this network makes use of both hidden layers, it does not solve the Four Quadrant problem in either of the regular fashions described above. Miller et al. also report that the GA consistently created a link directly from the input unit to the output unit, a feature that was allowed by their network architecture description but not anticipated by them. It is important to note, however, that some of these solutions existed in the random initial population with which the GA began, and the GA converged on a population of such individuals in only a few generations. The GA did not therefore play an important role in the discovery of these solutions, and any iterated restart of BP can be expected to have performed similarly.

Several comments should be made about binary wiring diagrams like these. First, experiments such as these impose an unfortunate asymmetry between the adaptation effected by the GA and that done by connectionist learning. Virtually all connectionist learning algorithms allow connections to come to have zero weight, making them act as if the connections were not there. Thus an existing connection can learn to have zero weight, but an

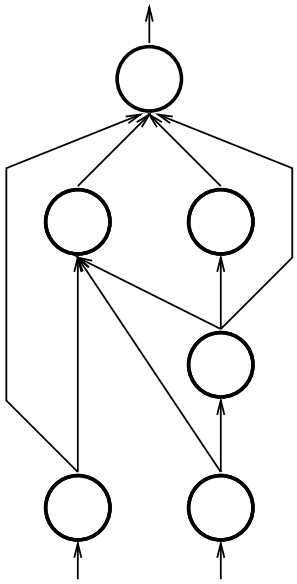


Figure 5: GA solution to Four-quadrant problem

absent connection cannot ever become non-zero. We should expect this bias to be exploited by any hybrid adaptive system that combines evolutionary and (connectionist) learning sub-systems.

Second, the binary specification of link presence or absence can easily be generalized to a wider range of constraints on network architectures. For example, Todd suggests that a ternary specification might specify that a link was absent, restricted to positive weights or restricted to negative weights [Todd, 1988]. The range of search allowed the connectionist learning procedure by the genetic network description can be progressively constrained in this fashion until, in the extreme, the GA is specifying each weight exactly. Conversely, the process of dynamic parameter encoding (DPE) can be used to focus the GA's search on those regions with least variability, so that *a priori* divisions of the search between GA and gradient procedures come to be reconsidered [Schraudolph and Belew, 1990].

Third, as soon as the goal of our hybrid algorithm is changed from finding the *weights* for a net that can do the best job, to finding an *architecture and also weights* for that architecture that can do the best job, our search criterion must change correspondingly. More specifically, if the GA is to

find good architectures, the function it optimizes must include not only a measure of error (e.g., MSE), but also a measure of the *complexity* of the network. Otherwise, if there is no (fitness) penalty paid for having more links, for example, there will be no adaptive pressure to use more parsimonious representations. The absense of any such complexity term in the Four Quadrant experiments of Miller et al. may account for the fact that the GA found neither the "fat" nor "skinny" solution, but something with (potentially redundant) direct links from input to output; see Section 6.

Finally, wiring diagrams do not avoid the "underdetermined architecture" problems described in the last section. The problem, at least in part, is that the relationship between genotype and phenotype is still inappropriate: features of the network that are inconsequential to its computation (e.g., the indexing of the hidden units) are reflected by radically different genetic descriptions. And so we are pushed another step away from complete network specification and towards the interposition of a developmental process between genotype and phenotype.

3.4 Developmental programs

The GA is generally cast as a function optimizer, with the GA manipulating values of $\underline{x}$ in order to optimize some function $f(\underline{x})$. One critical aspect of biological evolution that is missed in this formulation is that the space of *genotypes* manipulated by genetics is only indirectly related to the space of *phenotypes* which are evaluated by the environment. The process relating genotype to phenotype is, of course, *ontogenetic development.*

The developmental process is an extraordinarily complicated adaptive system in its own right [Purves, 1988], and attempting to incorporate it within the already complicated hybrids being considered here is problematic.[5] But the incorporation of a developmental interpreter means that the GA can be allowed to search through representations for which it is more well-suited than those derived directly from networks. Just which developmental model will prove most satisfactory in the context of evolution/development/learning hybrids is still an open and important question, but there are a few promising leads.

[5] For example, the developmental interpreter should, by rights, itself be specified on the genome.

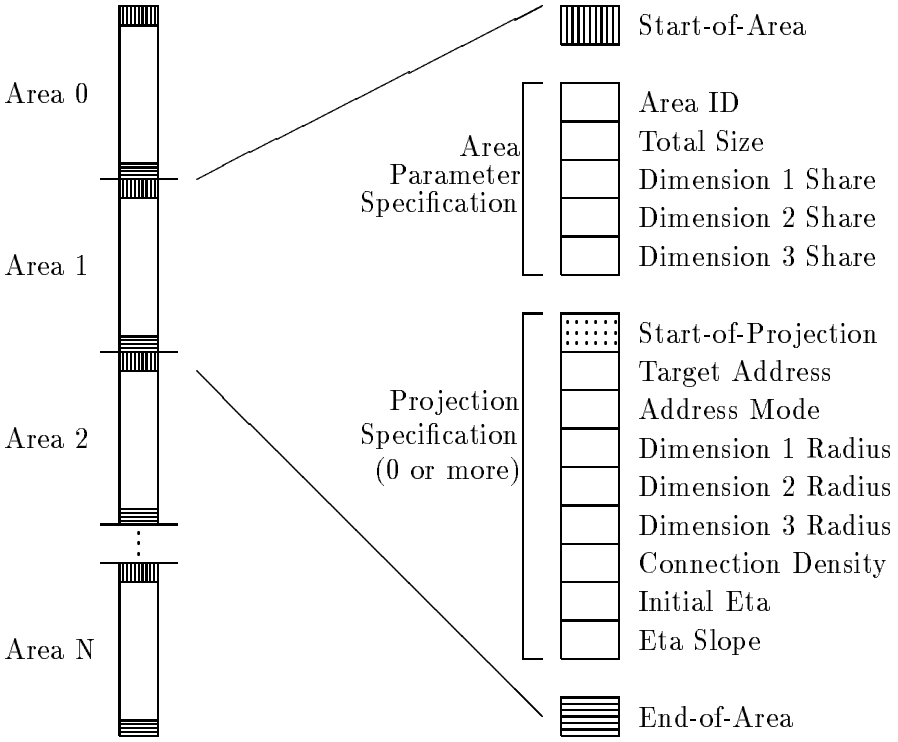


Figure 6: Harp blueprint

Harp et al. describe a very interesting model in which the GA searches for a network "blueprint" [Harp et al., 1989]; see Figure 6. The description of neural networks in terms of "areas" (i.e., sets of units with varying spatial extent), and "projections" (i.e., sets of edges randomly connecting units from one area to another) certainly seems to capture much of the architectural regularity of nervous systems in vertebrates.

These experiments are consistent with only the most basic features of the corresponding biological systems, and we intend to explore more satisfactory models. Purves outlines the basic features of a more elaborate cell adhesion model of neural development [Purves, 1988]; Edelman also desribes an elaborate but idiosyncratic model [Edelman, 1987]. The details of retina-to-optic tectum mappings have been described by Cowan [Cowan and Friedman, 1990]. Stork has used a developmental model with the GA to show how evolutionary "pre-adaptations" may be responsible for

certain anomolous neural connections in the modern crayfish [Stork et al., 1990].

The range of connectionist network representations for the GA surveyed in this section still leaves much to be explored. Further, the GA can be used with connectionist networks in other ways than specification of network architecture. The next two sections describe experiments that use the GA to control other, non-architectural parameters of connectionist learning systems.

4 Tuning BP parameters

The central experiments of Section 5 use the GA to find good initial weights from which a gradient descent procedure like back propagation (BP) can reliably converge on a solution. While these investigations are largely orthogonal to the use of the GA for the kind of network architectural definition described in the last section, the well known *symmetry* problem[6] was used because, like Miller et al's Four Quadrant problem (cf. Section 3.3), it is known to permit two distinctly different network solutions [Minsky and Papert, 1988, p. 252-253]. For the experiments reported here we used the six-bit version of this problem. A three-layer feedforward network with six input units, six hidden units, and one output unit was specified.

Before beginning these experiments, it was necessary to set the learning rate (η) and momentum (α) parameters of BP. These parameters are known to be strongly coupled, dependent on characteristics of the problem being solved, and critical to the successful convergence of the learning procedure. As a result, finding good values for η and α is more art than science and generally a matter for trial-and-error search. Because the GA has often had success at strongly non-linear function optimization problems like this, we began by using the GA to find good values for η and α. The ranges

$$0 \leq \eta \leq 8;\ 0 \leq \alpha \leq 1.0$$

were explored, and each parameter was allocated 10 bits. This unusually large range for η was used because preliminary experimentation with more

[6] Given a binary vector of length $2N$, the net is to produce a value of unity on its single output unit iff, for all input units $I_i = I_{2N-i+1}, i = 1, ..., N$.

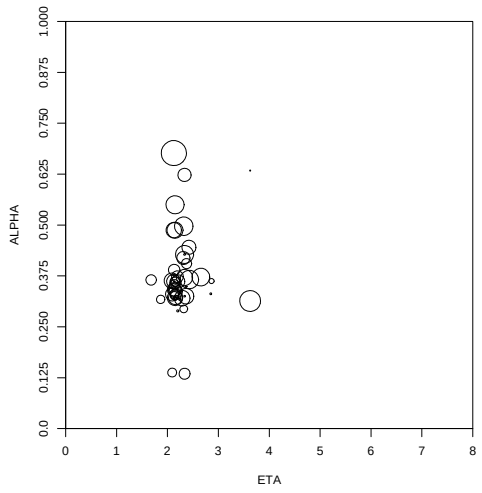


Figure 7: Finding η and α with the GA

conservative values (e.g., $\eta \leq 2, \eta \leq 4$) consistently resulted in the GA converging on the maximum value in that range.

With the GA selecting values for these two parameters for each individual in a population of size 50, an otherwise standard BP simulation was trained for 200 epochs and its mean squared error (MSE) at the end of this traing was used as the individual's fitness.

Figure 7 shows the results after 200 generations, with the diameter of each circle indicating the reciprocal of the MSE (i.e., larger circles mean better performance) for each individual in the last generation. Conventional wisdom calls for $\eta = 0.1, \alpha = 0.2$, but the GA consistently converged on values of $\eta \approx 2.5$! This means that BP is moving rapidly along the error surface. We conjecture that these values depend on the number of learning epochs ($\mathcal{T}$) we allowed each BP optimization before using the net's MSE value. We used $\mathcal{T} = 200$ epochs, which is an extremely short training

period for the six-bit symmetry problem[7]. Networks using more "patient" values for η and α were unable to solve the problem in the time alloted, and only those that "went for broke" were successful. It seems likely that more conservative values for η and α would be found if more trials were allowed, and even more radical values might be found if this number were reduced.[8]

Note also that both large and small diameter circles are sometimes virtually concentric. This means that while the GA consistently converged on fairly narrow ranges for η and α, there is still high variability in the fitnesses of individuals with very similar parameters. This is because the BP simulations have a highly stochastic element, viz., the random initial weights assigned. The values for $\eta = 2.5, \alpha = 0.33$ were robust enough under varying initial weights that they could be exploited by the GA, but the selection of good initial weights is still critical. The identification of these good initial values is the major focus of our next experiments.

5 Sampling and search

The use of the GA to tune BP parameters has proven practically useful, but there is nothing terribly profound about this type of hybrid. The GA is doing function optimization over a set of parameters in the same way that GAs have been used since DeJong [DeJong, 1980]. A more important combination of these technologies arises from the observation that *local search performed by back propagation and other gradient descent procedures is well complemented by the global sampling performed by the GA*; consider Figure 8. Gradient descent procedures all sample some characteristics of their local neighborhood to determine a direction in which the search is to proceed. Sophisticated techniques for gradient descent (of which BP is only one) efficiently combine characteristics of the local neighborhood to form a good next guess. But, depending on characteristics of the objective function being searched, this local information may be misleading as to the location of

[7] Randomly restarted BP runs using $\eta = 0.1, \alpha = 0.2$ reliably converge to the solution in approximately 4000 epochs.

[8] It should be noted that the use of these high, quick parameters for BP seem to depend critically, at least in the six-bit symmetry problem, on the order in which training instances are presented. Our simulator allows exemplars to be presented in: sequential order, random order with replacement, random order without replacement, or "batch" training. Because we have had most success on the six-bit symmetry problem using *random with replacement* ordering, these experiments were run with this option.

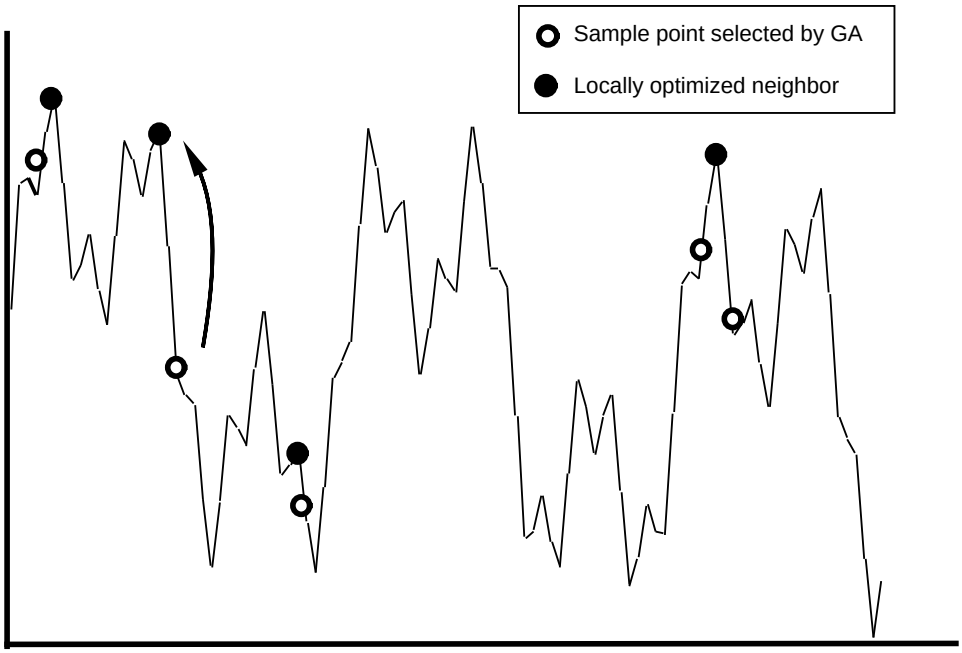


Figure 8: Sampling and search

the actual optimum. In particular, gradient descent procedures are known to be subject to local minima. Sampling techniques like the GA, on the other hand, are effective because they ensure broad coverage over the entire domain. The GA works by collecting information from the early and virtually uniform sampling of early generations, and then using this information to guide subsequent sampling towards particularly promising regions. The selection of a new element of the domain to evaluate therefore exploits global information from across the domain. Unfortunately, information in the immediate neighborhood of each of these samples plays no role in subsequent GA sampling, meaning the algorithm can come frustratingly close to the solution without actually finding it.

Combining the GA's global sampling with BP's local searching therefore seems an extremely natural and promising form of hybrid. It can be compared to the simple process of hill climbing with restart, with the key advantage of the GA being that it promises to sample in a much better than random fashion [Ackley, 1987,Goldberg, 1989].

When placed in the context of connectionist networks, the strategy just expressed suggests that the GA be used to create "seeds": starting points from which a connectionist search proceeds. In connectionist learning terms, this corresponds to using the GA to prescribe the *initial weights*, $\underline{\mathbf{W}}(\mathbf{0})$, on a network's links. A schematic view of this hybrid construction is shown in Figure 9. The GA selects an initial weight vector for each individual in a population, each is allowed to learn with BP for some number of trials, and the error rate at which it is performing at this time is considered to be the fitness of that individual.

Thus the GA will sample those regions of weight space from which it is reliably possible to reach good function values via gradient descent. There is a pleasing symmetry to this search, in that the best initial weight vector (found by the GA) is obviously the same as the final weight vector (found by BP); the two algorithms' solutions are interchangable in this respect. It is important to note, however, that the two algorithms are coupled in this symmetric search only if the range of initial weight values being explored by the GA is coextensive with the domain of solution weight vectors ultimately discovered by the gradient descent procedure.

In practice, these two sets are often quite different. For example, the PDP volumes recommend "... small random weights" [Rumelhart et al., 1986, p.

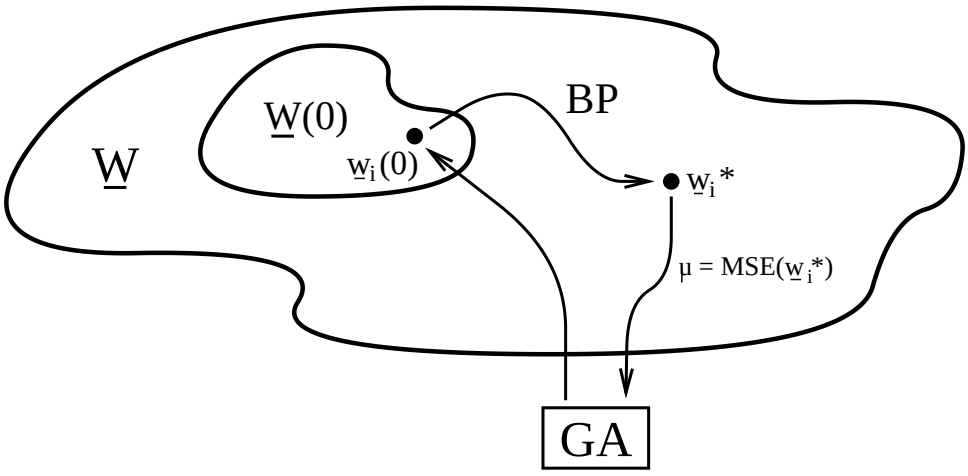


Figure 9: Selecting $\underline{\mathbf{W}}(\mathbf{0})$

330]; Miyata has operationalized this as "... uniformly distributed random numbers between $\pm\frac{1}{2}$" [Miyata, 1987]. Final weights, on the other hand, can be widely distributed, and often fall outside this initial distrution [Hanson and Burr, 1990].

The distinction between starting and final points in weight space also complicates our characterization of just what the GA is looking for. While it is true the GA is seeking $\underline{\mathbf{W}}(\mathbf{0})$ that are "close to" the solutions ultimately found by a gradient technique, it is important to note that the relevant measure is not the natural (e.g., Euclidean) distance between initial and final weight vectors. Rather, good $\underline{\mathbf{W}}(\mathbf{0})$ found by the GA are close to good final solutions *with respect to the gradient procedure being used.* Figure 10 caricatures the search regions induced by two different gradient techniques, all begun from the same initial point in weight space. For a particular gradient technique, these regions can be characterized in terms of "isobars" requiring the same computational effort (e.g, BP training epochs). We can imagine error surfaces over which it takes a gradient procedure many iterations to move a short Euclidean distance, and the converse is also true. Further, the range of solutions "reachable" by a gradient procedure varies with the procedure being used; points that are easy to reach via BP may not be reachable via conjugate gradient techniques, and vice versa.

Finally, sampling procedures like the GA require that the gradient pro-

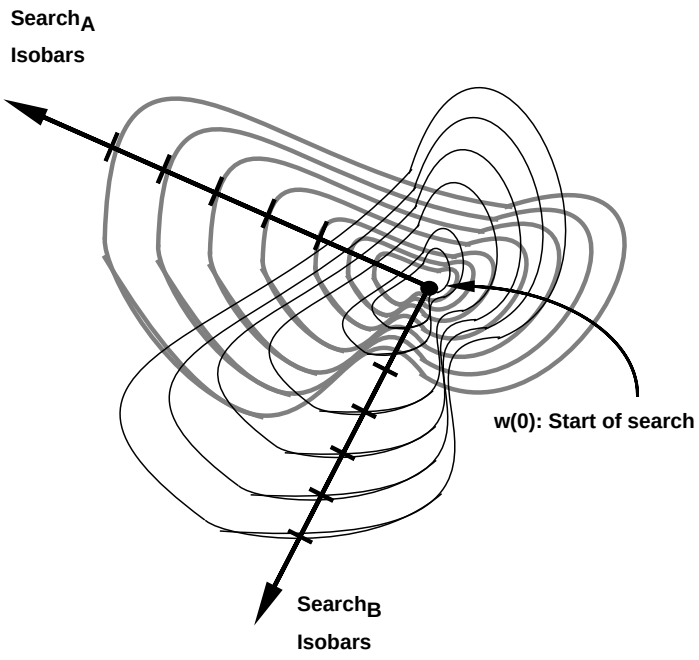


Figure 10: Gradient search isobars

Expt.	$\underline{\mathbf{W}}(\mathbf{0})$ *range*	*GA Encoding*	*Gradient proc.*	*Epochs*
1	$\pm 10^5$	Uniform	Conj. grad.	N/A
2	$\pm\frac{1}{2}$	Uniform	Back prop.	200
3	$\pm[10^{-2}, 5]$	Exponential	Back prop.	40

Table 1: $\underline{\mathbf{W}}(\mathbf{0})$ Experiments

Total Trials = 3000
Population Size = 50
Structure Length = 128
Crossover Rate = 0.600
Mutation Rate = 0.001
Generation Gap = 1.000
Scaling Window = 5
Max Gens w/o Eval = 2
Options = acel

Table 2: GA parameters

cedure *reliably* converge on a good solution. Recent results by Kolen and Pollack demonstrate that "BP is sensitive to initial conditions" (i.e., what we call $\underline{\mathbf{W}}(\mathbf{0})$) [Kolen and Pollack, 1990] , and so finding such reliable regions is non-trivial (cf. Section 5.4). Thus the goal of our search is somewhat different from most connectionist systems: we are interested in the *distribution* of good solutions rather than simply identifying some of these.

Three sets of experiments were performed to test the feasibility of using the GA as a source of $\underline{\mathbf{W}}(\mathbf{0})$ seeds for gradient techniques that then did their best to optimize further; these are summaraized in Table 1. In the first set a conjugate gradient (CG) method was used, and the range of $\underline{\mathbf{W}}(\mathbf{0})$ explored by the GA was made very large, effectively coextensive with that of the CG procedure. In the second set of simulations, BP was used and the GA was allowed to explore within the more typical range $\pm\frac{1}{2}$. Finally, an expanded range was searched by the GA, but an exponential encoding was used that allowed particularly refined searching of small $\underline{\mathbf{W}}(\mathbf{0})$ values. Basic parameters of the GA were also kept constant across the three experiments; see Table 2. Our group has developed a sophisticated GA simulator, Genesis-UCSD, and it was used for all of the experiments reported here; Section 6.1 describes a recent extension of this simulator that allows it to run across a distributed network of processors.

5.1 Experiment 1: Conjugate gradient over closed domains

To ensure the closure properties between GA and gradient search procedures mentioned above, the range of initial weights were allowed to run between $\pm 10^5$. Eight bits were allocated to represent each weight. Thus the GA was able to specify an initial weight with extremely poor precision: Each allele value corresponds to a range of approximately 800 (see Section 3 for details). Conjugate gradient (CG) optimization was used in these experiments because it is known to converge to solutions more quickly and reliably than heuristic second-order optimization techniques like BP (with a momentum term) [Barnard and Cole, 1989][9]. The GA was used to create an initial population of $\underline{\mathbf{W}}(\mathbf{0})$ vectors, CG optimized each of these, the MSE of each result was used for the individuals' fitnesses, the GA used these to produce a new population of $\underline{\mathbf{W}}(\mathbf{0})$ vectors, and the cycle repeated itself.

Figure 11 summarizes the basic results of these experiments. The average mean squared error (MSE) of the GA+CG hybrid is plotted on a logarithmic scale, as a function of generation; this curve is labelled **GA+CG(avg)**. To put the results of this hybrid method in perspective, it is appropriate to compare them to use of GA and CG methods used in isolation. For comparison with the CG-alone method, multiple randomly restarted iterations of CG were performed an equivalent number of times and this average baseline is labeled **CG (avg)**. The comparison of averages is appropriate given our interest in *expected* performance, but the simple average does blur information about the underlying distribution. Also, in most applications, the *minimum* of multiple restarts would be used. To facilitate this comparision, multiple randomly restarted iterations of CG were performed an equivalent number of times and the minimum of these is drawn as a baseline labeled **CG (min)**

Alternatively, the hybrid can be compared to search by the GA alone wherein initial $\underline{\mathbf{W}}(\mathbf{0})$ vectors selected by the GA were evaluated without modification by CG. In both cases, the hybrid approach did significantly better than either the gradient technique or the GA used in isolation. Thus, on average, the hybrid of GA+CG can solve the problem more effectively than either a randomly started CG search or GA sampling uninformed by

[9]The code implementing the CG method was obtained from the OPT package of Barnard and Cole. This code required extensive modification to allow it to be used for these problems, and to fix a serious bug it contained.

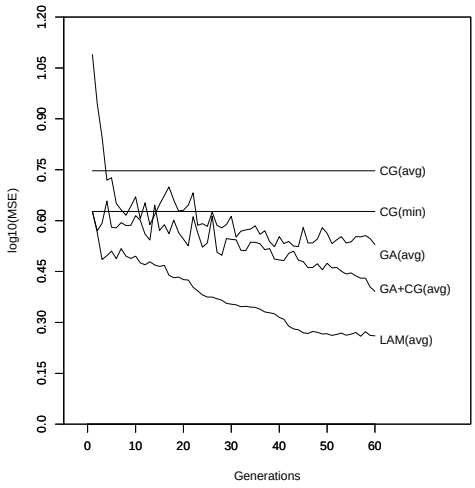


Figure 11: GA+CG hybrid

some gradient search.

At the end of 60 generations the GA+CG hybrid had converged significantly[10]. The solutions it found were odd, in that large magnitude weights were used, often greater than $\pm 10^4$. This seems to be due to two factors. First, the GA will sample large $\underline{\mathbf{W}}(\mathbf{0})$ weights more often than small ones, simply because there are so many more of them. Uniformly dividing a large weight range into a finite set of regions results in the low end of the dynamic range being grossly under-represented. (This suggests non-uniform, *exponential* encodings; see Section 5.3.) Second, the inclusion of even one large weight in the inner product summation performed by connectionist units is enough to push that unit's activity into the asymptotic range of its squashing function, effectively drowning out the contribution of any smaller weights. More precisely, the derivative of the sigmoidal squashing function is near zero for large weights, meaning that almost no error information is available for BP. The result of these two factors is that the solutions found by these experiments were poor in absolute terms.[11] (BP can find perfect solutions to six-bit symmetry, at least if given enough trials; see Section 6.)

Our picture of the range of good $\underline{\mathbf{W}}(\mathbf{0})$ values now looks like a "donut," with both lower and upper bounds; see Figure 12 If $\underline{\mathbf{W}}(\mathbf{0})$ is too near the origin, the network is unable to break the symmetries of its hidden units so that they all attempt to do the same job, and the network will remain on the "local maximum" of the origin originally described by Rumelhart et al.[Rumelhart et al., 1986, p. 330]. But if $\underline{\mathbf{W}}(\mathbf{0})$ is too large, the net is drawn to solutions with large weights that grow without bound. Worse, any *one* large weight can push a unit into the asymptotic region of its sigmoidal squashing function, effectively masking any error signal. For these reasons, the remaining two experiments of this report restrict the GA's search for good $\underline{\mathbf{W}}(\mathbf{0})$ values to a limited range.[12]

Before discussing these other experiments, however, another interesting

[10] 137 out of 384 alleles converged to at least 90%; Bias = 0.888.

[11] Another possible explanation is that the CG method was used inappropriately. Conjugate gradient techniques work well for moving quickly to the bottom of an attractor basin, but they may not be the best way to find such basins in the first place. This suggests that (yet another!) hybrid method would use BP for a few iterations, to get into a good attractor basin, and then invoke CG to finish mimimization. We are exploring this technique.

[12] In a forthcoming report, we investigate a theoretic characterization of the donut of appropriate $\underline{\mathbf{W}}(\mathbf{0})$ values.

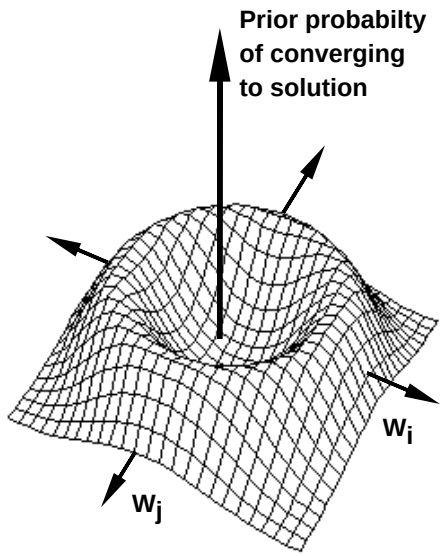


Figure 12: "Donut" of good $\underline{\mathbf{W}}(\mathbf{0})$

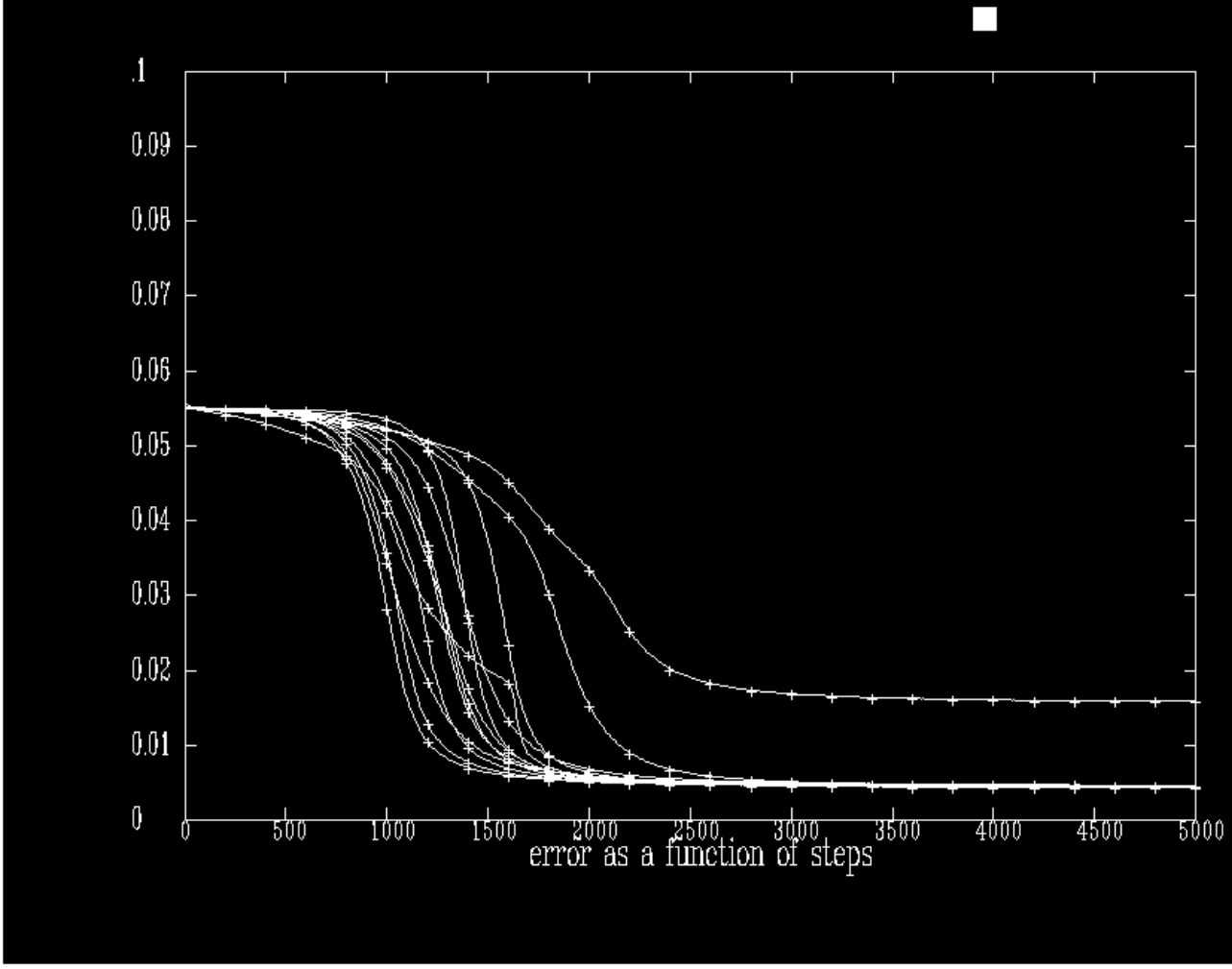
comparison is possible when the GA and gradient methods share coextensive search spaces. The GA+CG hybrid can be compared to a third technique that might be called a "Lamarckian" algorithm: the solution found by CG is remapped into the genetic encoding and this new specification is returned to the GA population in place of the original $\underline{\mathbf{W}}(\mathbf{0})$. That is, instead of giving the GA only the information that some individuals are close (in the gradient search sense) to a good solution, we do our best to "reverse transcribe" the solution actually found back into its genomic correlate, and give this to the GA to act as a new, genetically engineered parent. At the end of the runs shown in Figure 11 this Lamarkian variant appears to do only as well as the standard GA+CG hybrid, but in longer runs the Lamarkian algorithm does significantly better. Note that again the ability to invert from learned solutions back into their genetic correlates depends on these two domains being coextensive. In fact, even in these experiments in which the range of $\underline{\mathbf{W}}(\mathbf{0})$ was extended far beyond normal, the CG often moved to values outside the range of $\pm 10^5$; in this case, the genetic value was given its maximum or minimum value. This is only one, simple example of the difficulty in inverting the results of learning back into their genetic correlates; see Section 7.

5.2 Experiment 2: BP and limited $\underline{\mathbf{W}}(\mathbf{0})$ range

With the exception of the folk-wisdom (mentioned in Section 5) that initial weights should be small ($\pm\frac{1}{2}$) and random, little is known about the selection of good initial weights. Empirically, it has been widely observed that the performance of BP is highly variable with respect to the choice of $\underline{\mathbf{W}}(\mathbf{0})$. Figure 13 is typical. This shows 10 standard[13] runs of BP varying only in their initial values for $\underline{\mathbf{W}}(\mathbf{0})$. All we can say is that sometimes BP works and sometimes it doesn't; this is far from satisfying.

The question to be investigated here is whether the GA can find regions of the $\underline{\mathbf{W}}(\mathbf{0})$ space that reliably lead to good (in the sense of low MSE) networks. The basic construction of the GA+BP hybrid algorithm is the same as the GA+CG hybrid above: The GA was used to create an initial population of $\underline{\mathbf{W}}(\mathbf{0})$ vectors, BP was then used to optimize each of these, the MSE of each result was used for the individual's fitness, the GA used

[13] Miyata's SunNet simulator [Miyata, 1987] was used on the problem of six-bit symmetry with six hidden units; $\eta = 0.1, \alpha = 0.2$.


.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
error as a function of steps

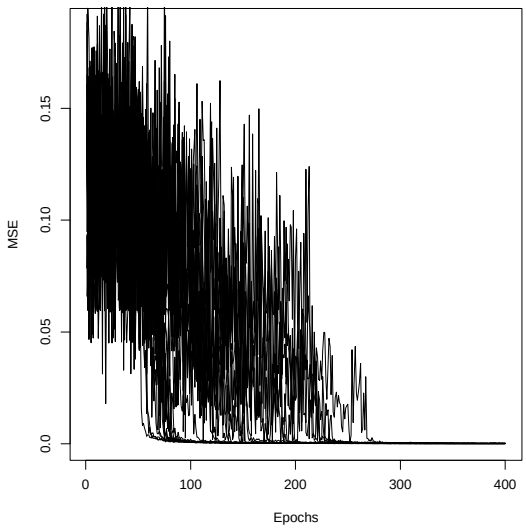


Figure 14: GA+BP Initial Population

these to produce a new population of $\underline{\mathbf{W}}(\mathbf{0})$ vectors, and the cycle repeats itself. However, for the reasons discussed in the previous section, the range of $\underline{\mathbf{W}}(\mathbf{0})$ explored by the GA was limited to the standard $\pm\frac{1}{2}$. Note also that an unusally high learning rate ($\eta = 2.5$) similar to those discovered in Section 4 was used in these BP simulations.[14] This rapid learning rate made it possible to give each BP network only 200 training epochs before judging its MSE fitness.

Figure 14 shows the learning curves for individuals in the initial GA population. Except that the curves are much noiser (due to the high η value used), the same high variability of Figure 13 is exhibited. And this is to be expected since the GA's initial population is also picked in a uniformly random fashion. Figure 15 shows that after 200 generations the GA was able to find a region of $\underline{\mathbf{W}}(\mathbf{0})$ space from which BP can reliably converge on solutions.

As with conjugate gradient, the performance of the BP+GA hybrid can also be compared to the performance of BP and GA used independently;

[14] These experiments were done before the simulations of Section 4 were complete, and hence the values ($\eta = 2.5, \alpha = 0.0$) are less than optimal.

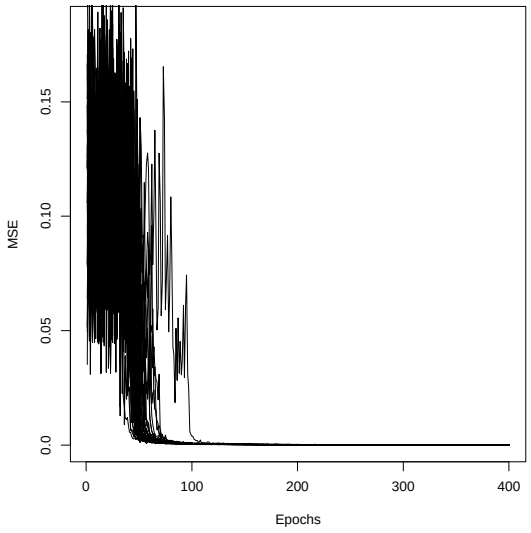


Figure 15: GA+BP Final Population

this comparision is shown in Figure 16. Here the generation average mean squared error (MSE) of the BP+GA hybrid is plotted on a logarithmic scale, as a function of generation and labelled **GA+BP(avg)**. The generation average of population in which GA was used to select $\underline{\mathbf{W}}(\mathbf{0})$ and the MSE of these individuals is taken immediately (i.e., without any BP learning) is labeled **GA(avg)**. For comparison with BP, two baselines corresponding to the average and minimum of an equal number of multiple random restarts of a BP simulation are shown as **BP(avg)** and **BP(min)**, resp.

There are several things worth noting in this comparison. First, the GA used by itself fares almost as well as the BP average. This is particularly striking given that the GA is only being allowed to sample in the original $\underline{\mathbf{W}}(\mathbf{0})$ space, $\pm\frac{1}{2}$. Second, use of the BP+GA hybrid creates a population of 50 individuals who, while very different from one another, have MSE performance almost identical with the best found by 5000 random restarts of BP alone. Finally, after only a few generations, the BP+GA hybrid is able to find strictly better individuals than could be found by 5000 independent BP runs, and ultimately finds a much better one. As with the conjugate gradient experiments of Section 5.1, the hybrid of GA+BP can reliably solve the problem more effectively than either a randomly started BP search or

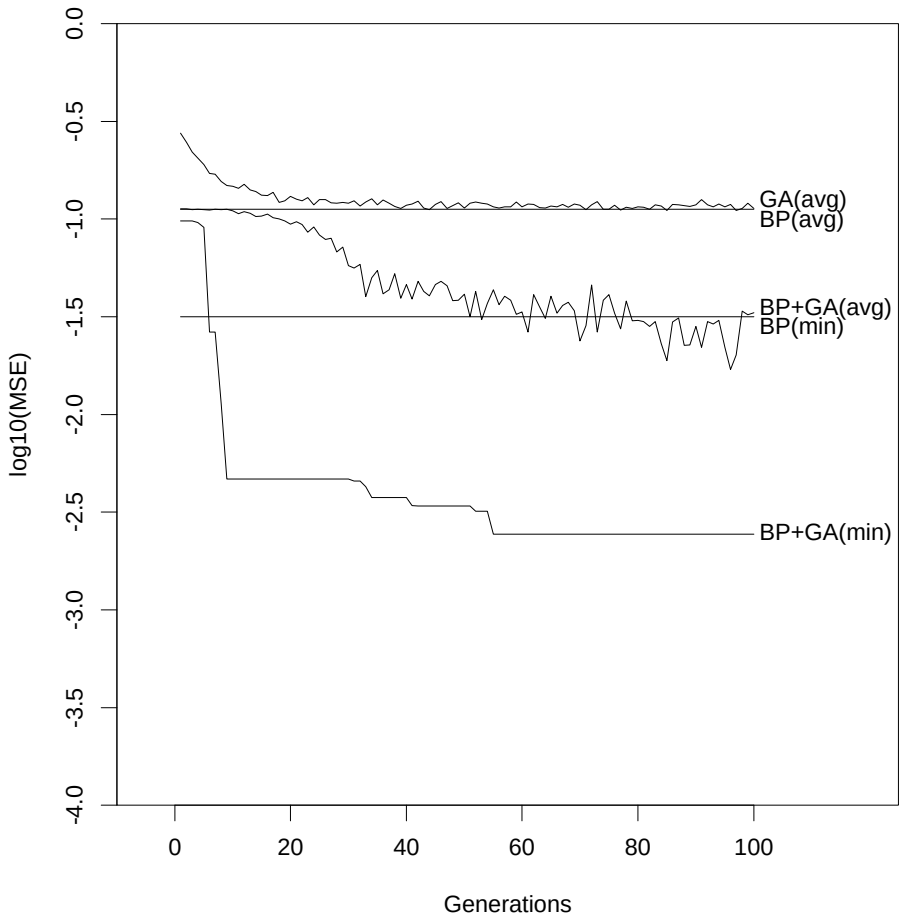


Figure 16: GA+BP hybrid

GA sampling without guidance by local information.

5.3 Experiment 3: BP over a closed domain

The results of the previous two sets of experiments appear to create a paradox: The GA should be allowed to sample over the same "closed" space of weights through which the gradient technique is to search, but if the genome is allowed to encode the entire range of weights, small $\underline{\mathbf{W}}(\mathbf{0})$ weights can be only sparsely represented, and the probability of the GA's sampling the fertile region in which *all* the weights are near zero is very small indeed. This paradox can be resolved, however, if the requirement that the $\underline{\mathbf{W}}(\mathbf{0})$ range be *uniformly* represented on the genome is relaxed. Real numbers are mapped onto the GA chromosome as before, but the final `TRANSFORM` stage (cf. Figure **??**) is changed from a linear transformation to an *exponential* one, so as to emphasize sampling of small weights.

Specification of an appropriate exponential sampling transform turns out to be a subtle issue that we are continuing to investigate. For these experiment our initial strategy was simply to select upper and lower bounds for $\underline{\mathbf{W}}(\mathbf{0})$, and the number of bits per weight. Based in part on Hanson and Burr's analysis of two large and well-studied networks [Hanson and Burr, 1990], we allowed $\underline{\mathbf{W}}(\mathbf{0})$ to range between approximately:

$$0.01 \leq | \underline{\mathbf{W}}(\mathbf{0}) | \leq 12$$

and allowed 10 bits/weight.

Using this encoding, the experiments of the last section combining GA and BP were repeated. The first observation was that this encoding was propitious in that it allowed extremely short BP training times. Figure 17 shows the results of combining the GA sampling process with only 40 epochs of BP training; as above, these results are compared to multiple, random BP runs and the use of GA without any BP search. With this short training, the best solution found by the hybrid GA+BP system was still better than that found by an equal number of randomly restarted BP solutions, but the difference in MSE's was less dramatic.[15]

[15] Still, the hybrid's solution solved the six-bit symmetry problem to criterion while the BP network did not.

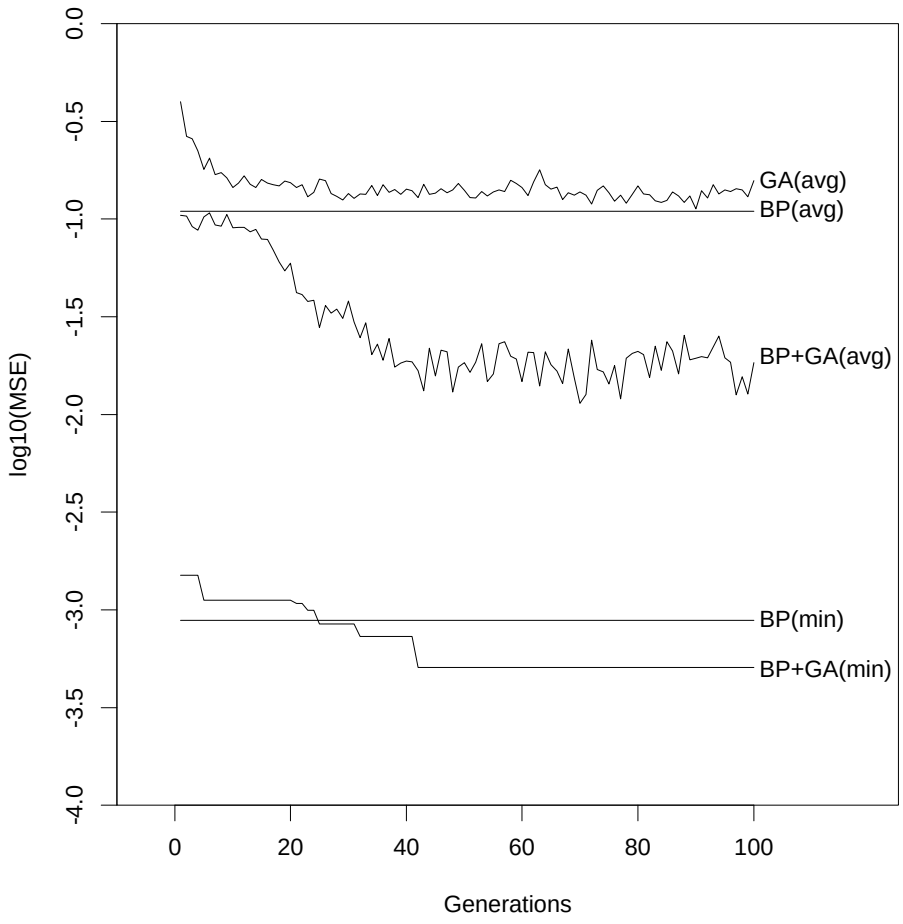


Figure 17: Exponential GA+BP hybrid

With longer training times (e.g., the 200 epochs used for the previous section's experiments), the problem became too easy for the GA: initial, random generations already contained many good solutions. We conjecture that constructing initial $\underline{\mathbf{W}}(\mathbf{0})$ vectors with many small weights but a few large ones — in effect much like the *final* weight distribution reported by Hanson and Burr — is enough to break hidden unit symmetries and still allow most error information past the sigmoid's derivative. When the training time is then reduced (down to 40 epochs!) to compensate for the facilatory effect of the exponential encoding, the GA is again able to find good regions of $\underline{\mathbf{W}}(\mathbf{0})$.

5.4 Sampling with the GA

The picture we have, then, is of a highly "textured" error surface, with the starting point of a gradient technique's search dramatically influencing its ability to get to satisfactory minima. Some basic bounds on a good $\underline{\mathbf{W}}(\mathbf{0})$ region can limit search to a "donut" around the origin; see Figure 12 and accompanying discussion. But even within this donut there is significant variability.

The experiments of Kolen and Pollack provide an interesting comparison with our own [Kolen and Pollack, 1990]. Among other experiments, they performed a Monte Carlo search through (what we call) $\underline{\mathbf{W}}(\mathbf{0})$ -space for the problem of 2-bit XOR with two hidden units. First, they echo our "donut" observation: "... the magnitude of the initial condition vector (in weight space) is a very significant parameter in convergence time variability." Second, they go further than characterizing the error surface as simply textured, to propse that they are "fractal-like." But the central difference between this work and our own is our use of the GA, effectively in place of their uniformly distributed Monte Carlo iteration. The moral Kolen and Pollack draw from their experiments is that since BP simulations are extremely sensitive to their initial $\underline{\mathbf{W}}(\mathbf{0})$ values, "... *the initial conditions for the network need to be precisely specified or filed in a public scientific database.* [Emphasis in the originial]." We believe our conclusion is more optimistic, and certainly less bureacratic: Use of the GA's non-random search allows us to judiciously sample $\underline{\mathbf{W}}(\mathbf{0})$ so as to identify regions from which we can *reliably* converge on good solutions, while simultaneously allowing us to cut training times by as much as two orders of magnitude.

The major finding from these experiments is that the use of the GA to select advantageous initial weights for the BP algorithm is effective. Section 3.4 outlined our current intuition that further progress with hybrid GA and gradient search procedures will require more sophisticated developmental programs, with the GA specifying broad constraints on patterns of neural connectivity rather than values for any one connection. But it is not that far-fetched to imagine that, at least in some primitive organisms' nervous systems, the genome does specify a coarse pattern of synaptic connections that is then fine-tuned during the organism's lifetime.

6 Computational complexity in GA/BP hybrids

On first blush, the idea of taking a compute-intensive procedure like BP and duplicating it $\mathcal{O}(100)$ times to form a population, and then using the GA to produce $\mathcal{O}(100)$ such generations seems profligate. And if we are well satisfied with the results of a single BP run, this analysis may be correct. But we have three good reasons to question this analysis. First, BP performance is known to be highly variable under stochastic variation. Consequently, many investigators already use some sort of "outer loop" of multiple, random restarts to improve their confidence in the solutions found. Section 5 argued that the GA's sampling is far superior to such random restarts. Second, there is currently great interest in more elaborate network topologies than the standard single, fully connected hidden layer. However, extending BP and other learning techniques to these new topologies has proven difficult. The recent experiments reported in Section 3.3, particularly those of Todd et al. and Harp et al., are indications, albeit preliminary ones, that the GA can be a useful tool for exploring these novel architectures.

Finally, most of the experiments reported here have changed the BP algorithm so that its training time is greatly reduced. As algorithm designers, our primary concern must be with the total time taken by the hybrid system. The time complexity of this system is simply the product of the number of generations the GA is run, times the size of the population of each generation times the training time taken by each individual:

$$TotalTime = Generations * PopulationSize * TrainingTime$$

Thus there is a direct tradeoff between the number of generations and the number of trials allocated each individual. Using the GA to produce 100

generations of 50 individuals multiplies the apparatent time complexity by 5000. But Section 5.2 and Section 5.3 report on experiments in which the training time was reduced (from 4000 epochs on 6-6-1 symmetry) by factors of 20 and 100, resp. to the same error criterion. Thus the use of faster learning rates and judicious sampling of $\underline{\mathbf{W}}(\mathbf{0})$, the 5000- fold increase in time can be cut to a factor of 50. When the greater assurance in the answers found through the GA's robust, global searching is considered, and compared to the $O(10)$ random restarts often done with BP simulations, our hybrid methods are very competitive.

Much of our current research focuses on a more theoretic basis for the hybridization of GA and gradient techniques. The time complexity issues just mentioned introduce an important new parameter for hybrid connectionist/GA systems like these, viz., how long each connectionist minimization procedure should be allowed to iterate before the value it has found is used to determine that network's fitness. So, are we better off using many GA generations of short-lived individuals, or allowing each individual to search for longer, so as to produce a perhaps more informative value for the GA?[16]

One view of the space-complexity issues in hybridization is suggested by Figure 18. Section 3.3 mentioned a few of the alternative encodings of connectionist networks onto the GA's string that are available. These alternatives can be ordered in terms of the number of free parameters being searched by the GA and gradient procedures. At one extreme we can use only the GA, fully representing in full precision each of the network's weights and leaving BP nothing to do; at the other extreme, BP could be used exclusively. Intermediate between these are hybrids that encode some constraints on the structure of the BP network into the GA's chromosome: a bit specifying a link's presence, several bits specifying allowed weight ranges, etc. When a developmental component is interposed between the GA and BP, complexity issues become even more complicated.

It will take time to understand these tradeoffs completely. As mentioned in Section 3.3, some measure of a network's complexity must augment the basic performance measure (e.g., MSE), or the GA cannot impose adaptive pressure towards more parsimonious solutions. Rumelhart (personal communication) has attempted to use BP itself to explore new architectures, by including additional terms in the criterion function for number of hidden

[16] In other words, whether 'tis nobler to "live fast and die young" or "live long and prosper"?

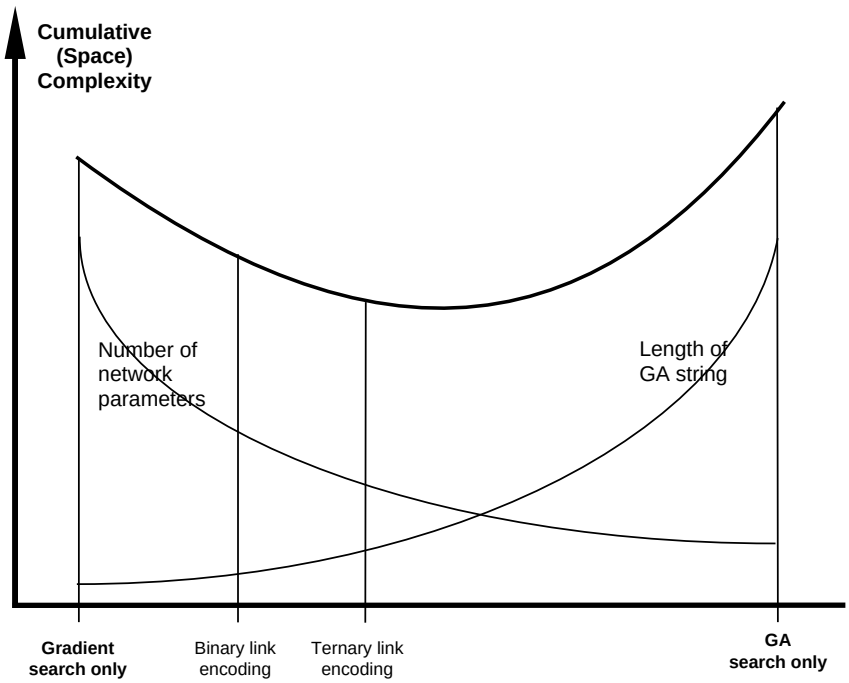


Figure 18: Cumulative Complexity

units, number of links, and even number of "distinct" weights used. Rissanen's "minimum description length" formalism provides a rigorous measure of a model's complexity [Rissanen, 1989], and Tenorio and Lee have made an initial attempt to apply this to connectionist network architectures [Tenorio and Lee, 1989]. Characterizing the description length of (binary) GA genomes promises to be more straight-forward, and the stage is then set for measuring the *cumulative* complexity of GA+network solutions in the manner originally suggested by Figure 18. But the time-complexity trade-offs mentioned above must also be incorporated, and interposing developmental programs is even more problematic. This is the core of our future theoretic work.

In the the interim, there is another aspect of our hybrid's computational character that we have already begun to exploit: GA/connectionist hybrids are eminently parallelizable.

6.1 Exploiting the parallelism of the GA

The availability of massively parallel computers, let alone large distributed networks of loosely coupled computers, has increased dramatically. Unfortunately, our ability to effectively utilize the vast computational power offered by these parallel systems has not kept pace. Mizell (personal communication) has singled out Monte Carlo-like computations as an example of "embarassingly parallelizable" applications (i.e., so naturally parallel that you get no credit for saying so) that have real utility and do tap the computational power of parallel machines and networks.

The basic "population" model underlying the GA (i.e., each individual in the population independently evaluating the objective function) makes it another candidate for parallelization, embarrased or not. We have exploited this feature of our hybrids in an extension of Grefenstette's GENESIS simulator developed by our group. The basic architecture of this Distributed GA (DGA) design is shown in Figure 19; Grefenstette has called this a "syncronized master-slave" architecture for the GA [Grefenstette, 1981]. Assume that some number H of hosts are connected via a local area network.[17] A `Host Table` with the name and network addresses of these hosts is constructed, a `BP Server` program is initialized on each. Then, a `GA Client`

[17] For our simulations, this environment has been a mixture of Sun 3's and 4's and various Vaxen, connected via Ethernet using TCP/IP protocol and NFS.

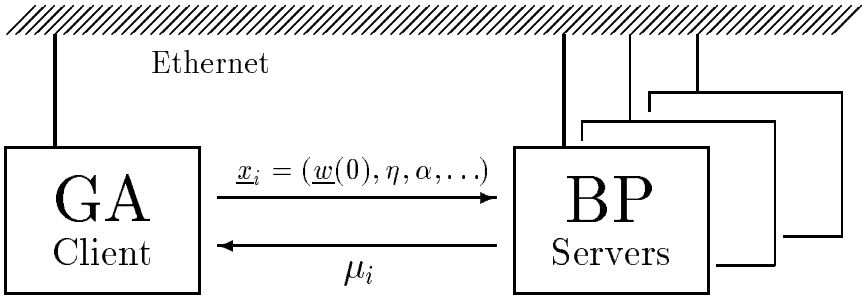


Figure 19: Distributed GA Model

program, which is very similar to the standard Genesis package, is begun on one host. This program creates an initial population with each individual corresponding to a particular parameterization of a BP simulation. These may be varied along a number of dimensions[18], although any one of our experiments has typically explored only one or two of these parameters.

In order to coordinate distribution of BP evaluations to the available servers, a `Population Table` is maintained, showing the status (`Evaluated`, `Being-Evaluated`, or `To-Be-Evaluated`) of each individual. Initially, a BP parameter packet is "dealt out" to each of the available BP servers. Typically, the population size exceeds the number of BP servers, so as evaluations are completed the `Population Table` is updated and the idle server is sent a new parameter packet to evaluate. This process repeats until the entire population has been dealt out at least once. After (an adaptively tuned) `Timeout Period`, any individuals still in the `Being-Evaluated` status are dealt out again, until all evaluations have been performed. The next generation's evaluations then commence.

This scheme works particularly well with a heterogeneous mix of computers of different processing speeds and user loads because the packets sent to and from the GA server are then randomized, avoiding bottlenecks; a slight amount of randomized delay has been incorporated for similar effect in a homogeneous environment. More recently, the `Host Table` has been augmented with statistics about how long it is taking each host to perform its BP evaluation. This information is used to dynamically alter the prior-

[18] Our simulator allows variation of: Number of hidden units, network wiring, η, α values, initial $\underline{\mathbf{W}}(\mathbf{0})$ weights, squashing functions, training regimes (random, sequential, permuted, batch), and training time.

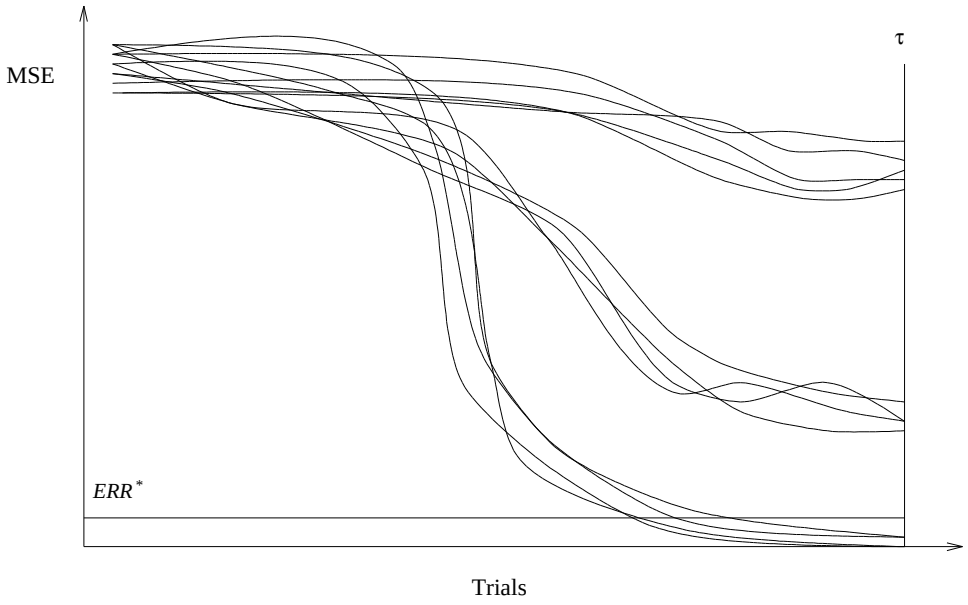


Figure 20: Learning curves

ity with which BP packets are assigned to the various hosts, for example avoiding heavily loaded or completely dead hosts.

The time-complexity issues mentioned above become very significant in the DGA design. For example, one undesirable bottleneck is created by the end of each generation, when all but the last one or two individuals have been evaluated and the rest of the processors must wait; Grefenstette calls this a "semi-syncronized master-slave" architecture [Grefenstette, 1981]. In general, with populations of reasonable size it is unlikely that these last few evaluations are critical, and so it seems reasonable to relax the constraint of a rigidly synchronized generational structure. Similarly, we note that populations of BP learning curves typically look something like those portrayed in Figure 20, with some simulations terminating because they have achieved the error criterion Err^*, and others terminating at a maximum number of training trials $\mathcal{T}$.[19] Ideally, it should be possible for the GA to advantageously manipulate both Err^* and $\mathcal{T}$, for example giving individuals who reach the error criteria quickly a high fitness value, or interupting slow evaluations if the rest of the population is done and has found good

[19] We are grateful to Peter Todd for suggesting this view.

values. The interupted processor is then freed to begin work on a potentially more informative evaluation. Finding an appropriate balance between the *exploitation* of quickly derived solutions versus the *exploration* of more time-consuming ones is a matter requiring further investigation.

7 Conclusion

This paper has reported on a series of experiments combining two popular classes of adaptive algorithms, Genetic Algorithms and connectionist networks. A survey of a wide range of potential mappings of connectionist networks onto the GA genome has convinced us that the most desirable representations for the GA can be expected to be quite removed from the most obvious network representations (Section 3). Our own work is pursuing biologically plausible ontogenic models that create virtual independence between the space of genomes being searched by the GA and the weight space of the connectionist networks (Section 3.4). We have demonstrated that the GA can be successfully used to tune parameters for the back propagation algorithm, at least for the problems we have investigated (Section 4). A much more substantial class of hybrids uses the GA to *globally sample* the space of initial network weights — $\underline{\mathbf{W}}(\mathbf{0})$ — from which connectionist gradient descent procedures can *locally search* (Section 5). These initial experiments have allowed us to focus several basic theoretic questions arising when GA and connectionist techniques are combined (Section 6), and to develop a distributed version of the GA that exploits the "embarrasingly" parellel nature of GA/connectionist hybrids.

Throughout, our experiments and discussion have remained in the province of computer science, concerned with issues of algorithmic design rather than the natural (genetic, neural, ontogenic) origin of these algorithms. We close with two observations about the computation being performed by these natural systems.

The first observation concerns a fundamental incongruity between the teleologies of biological systems and the artificial ones we run on computers. Much of Section 6 was concerned with refuting objections that wrapping the GA around BP was a waste of (computational) cycles. In our artificial algorithms it is appropriate to worry that a cycle used (for example) to produce a new generation might be better spent allowing an indivudal

another training epoch. That is, the total number of cycles is considered a *conserved quantity.* But it is not at all clear that biological populations are burdened by any such constraint. In fact, Evolution seems truly profligate with its "cycles," creating as many redundantly exploring individuals as an environment's "carrying capacity" will allow, and allowing each to live as long as mortality allows. As massively parallel computers become available, using a similarly profligate strategy of redundant search may become more sensible in algorithmic design as well (e.g., Hillis' search for efficient sort routines [Hillis, 1990]).

Second, a computational view provides new insight into the exploitation of individuals' learning and the evolution of a species. Despite the intuitive appeal of a theory that would allow individual learning to favorably influence evolution, the biology of genetic reproduction explicitly rules out the *direct*, "Lamarckian" inheritance of acquired (e.g., learned) characteristics. Previous work has demonstrated that at least some of the desirable interactions between learning and evolution can be explained via *indirect* mechanisms, such as the "Baldwin Effect" [Belew, 1990].

Experiments reported here suggest another computational reason why direct Lamarckian inheritance cannot be possible. In particular, Section 5 discussed the complex relationship between the space of initial weights ($\underline{\mathbf{W}}(\mathbf{0})$) searched by the GA and the space of ultimate weights discovered by the gradient techniques of conjugate gradient or BP. But if $\underline{\mathbf{W}}(\mathbf{0})$ is a proper, small subset of the ultimate weight space (as it was in the experiments of Section 5.2), the solutions found by BP cannot simply be "reverse transcribed" back into the GA's genomic representation! More generally, as the relation between network architecture and genomic specification becomes more and more indirect (for example through the use of the developmental translation programs we advocate) the ability to *invert* this relation diminishes. That is, given a mature, successful individual, it becomes harder and harder to invert the mature *cognitive representation* responsible for the successful performance into his or her original *genetic representation.*

We conjecture further that it is in fact impossible for a system to inherit (at least some) acquired and critical characteristics, not because of biological "implementation details" with reverse transcription (for we know evolution to be terribly inventive), but because it is *computationally* impossible to encode in the *structural* genotype the results of *behavioral* experiments. It is our goal to use our increasing understanding of the computational properties

of biologically plausible algorithms like the GA and connectionist networks to cast this conjecture formally. In any case, it appears that features of information processing by natural systems and characteristics of artificial computation are again intimately entwined.

Acknowledgements

We gratefully acknowledge useful discussions of this work with Peter Todd and Jeffrey Miller. Thanks also to George Wittenberg and Susan Gruber for comments on an earlier draft of this manuscript. Finally we thank the Cognitive Computer Science Research Group of the CSE department at UCSD for generally fomenting this and much other work.

References

[Ackley, 1987] Ackley, D. H. (1987). *A connectionist machine for genetic hillclimbing.* Kluwer, Boston.

[Barnard and Cole, 1989] Barnard, E. and Cole, R. A. (1989). A neural-net training program based on conjugate-gradient optimization. Technical report, Dept. Computer Science and Engr., Oregon Grad. Ctr., Beaverton, OR.

[Belew, 1990] Belew, R. K. (1990). Evolution, learning and culture: computational metaphors for adaptive search. *Complex Systems*, 4(1).

[Bethke, 1981] Bethke, A. (1981). Genetic algorithms as function optimizers. Technical report, Logic of Computers Group, CCS Dept., University of Michigan, Ann Arbor, MI.

[Brady, 1985] Brady, R. M. (1985). Optimization strategies gleaned from nature. *Nature*, 317:804–806.

[Caruana and Schaffer, 1988] Caruana, R. and Schaffer, J. D. (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Proc. Fifth Intl. Conf. on Machine Learning.* Morgan Kaufmann.

[Cowan and Friedman, 1990] Cowan, J. D. and Friedman, A. E. (1990). Development and regeneration of eye-brain maps: A computational model. In *Advances in Neural Info. Proc. Systems 2.* Morgan Kaufmann.

[DeJong, 1980] DeJong, K. (1980). Adaptive system design: A genetic approach. *IEEE Transactions on Systems, Man, and Cybernetics.*

[Edelman, 1987] Edelman, G. (1987). *Neural Darwinism: The theory of neuronal group selection.* Basic Books, New York.

[Fogel et al., 1966] Fogel, L., Owens, A., and Walsh, M. (1966). *Artificial intelligence through simulated evolution.* Wiley, New York.

[Goldberg, 1989] Goldberg, D. (1989). *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, MA.

[Grefenstette, 1985] Grefenstette, J., editor (1985). *Proc. First Intl. Conf. on Genetic Algorithms and their applications.*

[Grefenstette, 1987] Grefenstette, J., editor (1987). *Proc. 2nd Intl. Conf. on Genetic Algorithms*. Lawrence Erlbaum.

[Grefenstette, 1981] Grefenstette, J. J. (1981). Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, Computer Science Dept., Vanderbilt Univ., Nashville, TN.

[Hanson and Burr, 1990] Hanson, S. and Burr, D. (1990). What connectionist models learn: Learning and representation in connectionist networks. *Behavioral and Brain Sciences*.

[Harp et al., 1989] Harp, S., Samad, T., and Guha, A. (1989). Towards the genetic synthesis of neural networks. In *Proc. Third Intl. Conf. on Genetic Algorithms*, San Mateo, CA. Morgan Kaufmann.

[Hillis, 1990] Hillis, W. D. (1990). Co-evolving parasites improve simulated evolution as an optimization procedure. In *Proc. Conf. on Emergent Computation*. MIT Press.

[Holland, 1975] Holland, J. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI.

[Huynen and Hogweg, 1989] Huynen, M. A. and Hogweg, P. (1989). Genetic algorithms and information accumulation during the evolution of gene regulation. In Schaffer, J. D., editor, *Proc. Third Intl. Conf. on Genetic Algorithms*, Washington, D.C. Morgan Kaufman.

[Kauffman and Smith, 1986] Kauffman, S. and Smith, R. G. (1986). Adaptive automata based on Darwinian selection. *Physica D*, 22.

[Kolen and Pollack, 1990] Kolen, J. F. and Pollack, J. B. (1990). Back propagation is sensitive to initial conditions. Technical Report 90-JK-BPSIC, CIS Dept., Ohio St. Univ., Columbus, OH.

[Merrill and Port, 1988] Merrill, J. W. L. and Port, R. F. (1988). A stochastic learning algorithm for neural networks. Technical Report 236, Dept. Linguistics and Computer Science, Indiana Univ., Bloomington, IN.

[Miller et al., 1989] Miller, G., Todd, P., and Hegde, S. (1989). Designing neural networks using genetic algorithms. In *Proc. Third Intl. Conf. on Genetic Algorithms*, San Mateo, CA. Morgan Kaufmann.

[Minsky and Papert, 1988] Minsky, M. and Papert, S. (1988). *Perceptrons (expanded edition)*. MIT Press, Cambridge, MA.

[Miyata, 1987] Miyata, Y. (1987). Sunnet version 5.2. Technical Report 8708, Inst. for Cognitive Science, UCSD, La Jolla, CA.

[Montana and Davis, 1989] Montana, D. J. and Davis, L. (1989). Training feedforward networks using genetic algorithms. In *Proc. IJCAI*. Morgan Kaufman.

[Offutt, 1989] Offutt, D. (1989). A reinforcement learning algorithm for training fully and bidirectionally-interconnected connectionist networks. (draft).

[Purves, 1988] Purves, D. (1988). *Body and brain: A trophic theory of neural connections*. Harvard Univ. Press, Cambridge, MA.

[Rissanen, 1989] Rissanen, J. (1989). Stochastic complexity in statistical inquiry. Technical Report RJ-6901, IBM Research Div., Yorktown Heights, NY.

[Rizki and Conrad, 1986] Rizki, M. and Conrad, M. (1986). Computing the theory of evolution. *Physica D*, 42:83–99.

[Rumelhart et al., 1986] Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representations by error propagation. In McClelland, J. and Rumelhart, D., editors, *Parallel distributed processing: Explorations in the microstructure of cognition*, chapter 8. Bradford Books, Cambridge, MA.

[Schaffer and Morishima, 1985] Schaffer, J. and Morishima, A. (1985). An adaptive crossover distribution mechanism for genetic algorithms. In Grefenstette, J., editor, *Proc. Intl. Conf. on genetic algorithms and their applications*.

[Schaffer, 1989] Schaffer, J. D., editor (1989). *Proc. Third Intl. Conf. on Genetic Algorithms*, Washington, D.C. Morgan Kaufman.

[Schraudolph and Belew, 1990] Schraudolph, N. N. and Belew, R. K. (1990). Dynamic parameter encoding for Genetic Algorithms. Technical Report LAUR 90-2795, Los Alamos National Laboratory - Ctr. for Nonlinear Studies, Los Alamos, NM.

[Stork et al., 1990] Stork, D. G., Walker, S., Burns, M., and Jackson, B. (1990). Preadaptation in neural circuits. In *Proc. IJCNN (Vol. 1)*, New York. IEEE.

[Tenorio and Lee, 1989] Tenorio, M. F. and Lee, W. (1989). Self-organizing neural network for optimized supervised learning. Technical report, School of Electrical Engr., Purdue Univ., W. Lafayette, IN.

[Todd, 1988] Todd, P. (1988). Evolutionary methods for connectionist architectures.

[Whitley and Hanson, 1989] Whitley, D. and Hanson, T. (1989). Optimizing neural networks using faster, more accurate genetic search. In Schaffer, J. D., editor, *Proc. Third Intl. Conf. on Genetic Algorithms*, Washington, D.C. Morgan Kaufman.