Training Neural Networks by means of Genetic Algorithms Working on Very Long Chromosomes

Peter G. Korning
Computer Science Department
Aarhus University
Ny Munkegade, Building 540
DK – 8000 Aarhus C
Denmark
Email: aragorn@daimi.aau.dk

"…there's nothing like millions of years of really frustrating trial and error to give a species moral fibre and, in some cases, backbone."

Reaper Man, Terry Pratchett

Abstract

In the neural network / genetic algorithm community, rather limited success in the training of neural networks by genetic algorithms has been reported. In a paper by Whitley (1991), he claims that, due to “the multiple representations problem”, genetic algorithms will not effectively be able to train multilayer perceptrons, whoes chromosomal representation of its weights exceeds 300 bit’s. In the following paper, by use of a “real-life problem”, known to be non-trivial, and by a comparison with “classic” neural net training methods, I will try to show, that the modest success of applying genetic algorithms to the training of perceptrons, is caused not so much by the “multiple representations problem” as by the fact that problem-specific knowledge available is often ignored, thus making the problem unnecessarily tough for the genetic algorithm to solve. Special success is obtained by the use of a new fitness function, which takes into account the fact, that the search performed by a genetic algorithm is holistic, and not local as is usually the case when perceptrons are trained by traditional methods.

1 Motivation

Genetic algorithms (GAs) are one of the most general search techniques known in computer science. They have been applied with much success over a broad spectrum of problems, especially since their rise in connection with the publication of David Goldberg's "classic" book on the subject from 89 [Goldberg89a]. Yet their success in relation to neural networks (NN) has been limited. For the design of network architecture they have been tried on, among other things, the notorious XOR problem (by e.g. [Whitley89], [Miller et al.89], [Romaniuk93], [Zhang & Myhlenbein93]) with theoretically interesting results following. But the technique seems to be so computationally heavy that only with difficulty can it be applied to "real-life problems" where it is outdistanced by methods like Scott Fahlman's "Cascade Correlation" [Fahlman90]. For the training of neural network's weights and thresholds their use is more obvious, yet their success to date have been modest. A number of papers on the subject have been published during the last four or five years; e.g. [Montana89] and [Caudell & Dolan89]. They all seem to have two things in common. Firstly analyses imply, that the achieved results in reality stem from a stochastic hillclimb, (for a description of various hillclimbing techniques se [Ackley87]), rather than from a GA's hyperplane sampling. This is first of all evident in exceptionally high mutation rates; see e.g. [Ronald & Schoenauer94]. Instead of being a background operator, it is in reality the mutation operator that scans the search space, guided by the GA's selectivity. Secondly, the chromosomes representing the coding of the weights of the network in question, are usually relatively short. Montana uses strings consisting of eleven real numbers, which must be assumed to be insufficient for setting "the building block hypothesis" to work. Caudell & Dolan use 288 bits; not much either.

The above conditions are summarized in a paper by Darrell Whitley [Whitley et al.91], in which it is concluded, that GAs are unsuited for the training of perceptrons, in cases where the coding of the weights consists of more than 300 bits. For this aim he proposes the use of a hybrid, a "genetic hillclimber", called GENITOR, which as the name implies is primarily based on mutation/selection.

My main purpose with this paper is to render probable that it is yet too early to abandon the idea of GAs as a general alternative to back-propagation (BP), when training multilayer perceptrons. It seems that the number 300 bits is of a size large enough to be non-trivial, but too small for the GA's hyperplane sampling to make itself felt. Below I will describe an experiment where the chromosome length is in the magnitude of 10000 bits; a couple of magnitudes longer than traditionally. The techniques I invent in this connection are generally applicable to NN training. Most importantly I use, as the first in GA training of NN, a fitness function which is *not* the least mean square function known from BP training (where it is used as error function). My new fitness function takes into account the qualitative nature of the GA's search.

Finally, I will focus on the generalization abilities of GA trained nets as this very important aspect, (indeed the very reason for using NN in the first place, on a given problem) is most often ignored in the literature of this field of research.

1.1 The Neural Nets' Task

My "real-life problem" consists in training a fully connected perceptron with 6 input nodes, a hidden layer containing 15 nodes, and one single output-unit, for the purpose of sorting corn in two categories: "wanted" (wheat) and "unwanted" (rye, barley and oats), see fig. 1. The inputs are obtained via a scanner and some simple

preprocessing.

FIGURE 1 HERE

The problem is non-trivial, one of the reasons being that overlaps exist between the different corn species, each of these consisting of a number of subspecies, measured on the basis of the six characteristics (length of grain, width, area ect.), the network is given as input. E.g., one subspecies of wheat looks more like rye, than it looks like any other subspecies of wheat. The problem was used as a main project for an introductionary course in neural computation for undergraduates at the University of Aarhus in the autumn of 93. Around 40 people competed in producing the best network. This makes the problem well suited for a GA/BP-comparison as the BP aspect was thoroughly investigated then, including various extensions of standard BP, such as Weight-decay, adaptive learning-rates, alternative cost-functions and constraints. I will return to this point shortly. The success criteria was:

"No. of correctly classified "wanted" grains" larger or equal to 85.0 % of all wanted grains

"No. of correctly classified "unwanted" grains" larger or equal to 99.0 % of all unwanted grains,

and the latter maximized. The grains are in fact used as seed corn, so a high level of purity must be obtained. Our (my team's) record in 93 was:

"No. of correctly classified "wanted" grains" = 87.5000 %

"No. of correctly classified "unwanted" grains" = 99.4444 %,

obtained after 1554 on-line training epochs, on a test set consisting of 2000 wanted and 4500 unwanted grains. This result will be used as a bench-mark in what follows. As I know that a good solution exists for the above neural net architecture, it becomes a question of the GA's (and my!) ability to (re)find it. What follows is a short

description of the various techniques tried in connection with standard BP and their relative success.

1.2 Alternative cost functions

Three alternatives to the standard quadratic error measure usually used as cost function in BP $\left(\zeta_i^\mu - O_i^\mu\right)^2$, where ζ_i^μ is the desired and O_i^μ the actual output for output node i on propagation of input pattern μ, were tried. Most success was obtained by the use of the Relative Entropy Function:

$$E = \sum_{i,\mu}\left[\tfrac{1}{2}\left(1+\zeta_i^\mu\right)\log\frac{1+\zeta_i^\mu}{1+O_i^\mu} + \tfrac{1}{2}\left(1-\zeta_i^\mu\right)\log\frac{1-\zeta_i^\mu}{1-O_i^\mu}\right]$$

It origins from information theory. See e.g. [Solla et al.88]. In effect it accelerates convergence on plains in the error landscape where the ordinary error function could get stuck, and decelerates progress on sharp bends of the cost surface, thus hopefully preventing oscillations around a minimum. My experiments did show a somewhat faster convergence, but not a better final result than what can be produced by the quadratic error measure. A middle course suggested by Fahlman [Fahlman89] is a new delta value for the output layer:

$$\delta_i^\mu = \left[g'\left(h_i^\mu\right) + \varepsilon\right]\left(\zeta_i^\mu - O_i^\mu\right)$$

"g" is the neurons' activation function. By way of ε, a minimum "speed" on the plains is assured, while the acceleration on the curves is kept. This works quite as well as the relative entropy function; an indication of the fact that it is the flat areas of the error landscape that constitute a problem in BP. But again there is no real improvement of the quality of the end result. The last alternative is also Fahlman's work. Ignoring the actual error function we use a new delta value for the back-propagation of the output layer:

$$\delta_i^\mu = \operatorname{arctanh}\tfrac{1}{2}\left(\zeta_i^\mu - O_i^\mu\right)$$

The idea being that when the difference between the expected and the actual output becomes large, we use a large step size. It performed as the previous function.

1.3 Weight decay

Here I started with a relatively high number of neurons, about 50, in the hidden layer, hoping that weight decay (WD) would remove non-productive weights by converging them towards zero. The rule employed was $w_{ij}^{new} = (1 - \varepsilon_{ij}) w_{ij}^{old}$. The strategy being that if BP is not actively trying to reinforce a given weight, the factor $(1 - \varepsilon_{ij})$ will send it towards zero (with exponential speed). The term:

$$\varepsilon_{ij} = \frac{\gamma\eta}{\left(1 + w_{ij}^2\right)^2}$$

insures that small weights will decay more rapidly than large weights. Otherwise we will discourage large weights, which is *not* our objective. If all weights relevant to a given neuron are removed, the neuron itself can be removed. In the end, one should be left with a network where only the relevant structures remain. WD sounds intuitively attractive, but the results were disappointing. Though a large interval of ε_{ij} values was tested, the expected effect did not materialize. All that happened was that all the perceptron's weights/thresholds were lowered with a factor that even appeared rather constant for a given trial. Lack of success using WD was a general experience among the approx. 40 participants engaged in classification of corn in 93. Only a single participant reported some success.

1.4 Constraints on BP

As a new thing, we tried to impose some constraints on BP's search of the error landscape, not unlike a kind of controlled simulated annealing. Our starting point was the following strategy: If a number of X epochs of BP have not improved the perceptron's

performance on a test set, then restore the old weights and try again in another direction; however, at most Y times. This is to prevent the BP from bouncing in a "hollow" that it takes more than X steps to get out of. If the latter is the case (i.e. we have tried Y times to get away from the hollow), X is reset to allow further steps away from the hollow to be taken, otherwise (i.e. the X epochs HAVE improved the performance) Y is reset. These constraints gave a significant speed-up of the learning process; typically a factor two or three. But again- no improvement on the final test set.

1.5 Interactive Learning-Rate

The following technique is in effect rather like the above-mentioned constraints, apart from the fact that it is exponential in time instead of linear. However, the effect is here obtained by an adjustment of BP's learning rate. We started out with a learning rate on 0.04 and halved it each 25th (on-line) epoch. For each 250th epoch we reset it to 0.04, to avoid rummaging about in a local minimum. It was by use of this technique that we obtained our record for classification of corn by BP:
"No. of correctly classified "wanted" grains" = 87.5000 %
"No. of correctly classified "unwanted" grains" = 99.4444 %
Interactive learning rate gives a certain speed-up for most runs. In return there is also a bigger ratio of erroneous runs compared to standard BP, meaning runs where nothing like an acceptable result is ever obtained. This is probably because the interactive learning rate very quickly drills down into the minima *that are present by virtue of the initial weights*. When the initial weights coordinate a bad area of the error landscape, we thus land on "rocky ground".

2 The Genetic Algorithm

In what follows, I will describe my GA from Davis & Steenstrup's five-component outline [Davis87]. The GA was

implemented in standard ANSI-C (gnu-c ver. 2.5.6), and run on a HP 705 “Snake” computer.

2.1 Chromosomal Representation of the Problem

I have chosen a binary representation of the chromosomes as this choice maximizes the number of building blocks per information unit. Most earlier attempts of GA training of NN use real value strings, meaning that crossover only takes place in the space between weights. This further means that the only way the separate weights are actually changed is by mutation. Such a strategy might not be taking full advantage of the GA’s processing capabilities.

A perceptron is coded as a concatenation of its weights. As thresholds (biases) in practice are calculated as weights associated to a neuron hardwired to the value -1.0, our perceptron (and thus our chromosomes) contains $(6+1)\cdot 15=105$ weights between the input layer and the hidden layer, and $(15+1)\cdot 1=16$ weights between the hidden layer and the output layer, making a total of 105 + 16 = 121 weights. Each of these are represented by four 29 bit integers, making $4\cdot 29 = 116$ bits available for each weight. These bits are uniformly mapped into the real-value interval [-16; 15] by the C program. At these limits, the neurons’ activation function is, for all practical purposes, completely saturated.

FIGURE 2 HERE

How many of the 116 bits that are actually used, depends of the precision the user requires. A chromosome thus maximally consists of $121\cdot 116 = 14036$ bits. See fig. 2. In the experimental results that follow, all 14036 bits are used when the genetic operators are applied (on the genotype). However, when the chromosomes are decoded to perceptrons (the phenotype) the numbers are rounded off to C’s ordinary “double” precision, 53 bits. Consequently I am,

with 121·53 = 6413 bits for the phenotype (and 14036 bits for the genotype), well on the other side of the 300 bits barrier found by Whitley experimentally.

But in what order should the separate weights be concatenated? Whitley et al. mention, as the primary source of the difficulties of using GAs for NN training, The Multiple Symmetric Representations Problem (MSRP). In each layer in a given perceptron, an arbitrary neuron and its incoming and outgoing weights, can switch places with any other neuron in the same layer, without changing the perceptron's functionality. It is symmetric. In our case it follows that there are 16!, i.e. approx. $2.1 \cdot 10^{13}$ possible permutations of an arbitrary solution! This means that the error landscape is extremely multimodal. One cannot rule out the possibility that a large amount of the processing power will be wasted on a futile oscillation between equally attractive weights. I have to a certain degree tried to solve this problem by coding the perceptron in such a way that a node in the hidden layer has its incoming weights distributed across most of the chromosome, instead of grouping the incoming weights of a neuron in the hidden layer together, as common knowledge would suggest. See fig. 3. In this way, the GA is to a certain degree forced to more or less "choose" one of the possible permutations early in a run, as a templet for future generations to work on.

FIGURE 3 HERE

Why is it so? Well, if we have a MSRP of size 16! it means that the potential "troublemaking" clusters of building blocks number 16 in a given solution. But with a coding, like the one in fig. 3, there are only 7 clusters of troublemakers (because there are 7 nodes in the input layer). Thus, if we believe in the building block hypothesis, it means that only 7! = 5040 of the possible 16!

permutations of a given solution should have a reasonable probability of being considered. This reduction from approx. 2 $2.1 \cdot 10^{13}$ to 5040 might be expected to boost the GA's performance somewhat.

2.2 The Initial Population of Solutions

I choose, inspired by Herz, Krogh & Palmer [Herz91], the initial weights at random, uniformly distributed in the interval plus/minus the inverse of the square root of the fan-in to the neuron in question. I.e. $\left[-\frac{1}{\sqrt{6}};\frac{1}{\sqrt{6}}\right]$and $\left[-\frac{1}{\sqrt{15}};\frac{1}{\sqrt{15}}\right]$ for the hidden layer and the output layer, respectively. Generally speaking, Gas are characterized by being less dependent on the initial values for success, than BP. The most important thing being that all possible alleles (i.e. "0" and "1") for a given locus in the chromosome are present. With a binary alphabet and a population size of 100 individuals, this will statistically "always" be the case (the probability being $1-2\cdot\left(\frac{1}{2}\right)^{100} \approx 1-1.6\cdot 10^{-30} \approx 1$). The initial population, generated as described above, by and large, turns out to consist almost entirely of individuals that either categorize all grains as being wheat or categorize all grains as being non-wheat. See section 3.6 for a discussion of the importance of the initialization.

2.3 Fitness Function, Selection Technique ect.

The raw fitness varies from experiment to experiment and is mentioned in the relevant sections. Common to all the different fitnesses is, however, the fact that they are all based on the chromosome in question being decoded, loaded into a perceptron, and propagated by a training set, consisting of the data of approx. 2000 grains. On the basis of the success of this classification procedure, the raw fitness is created. Instead of scaling the raw fitness, I have chosen to use Fitness Ranking [Baker85], [Davis89], [Whitley89]. The solutions are sorted by their raw fitness and

assigned a Reproductive Fitness according to their rank. I use an exponential fitness ranking with base 0.95; meaning that the best individual will receive reproductive fitness 0.95^0, the second best individual receives 0.95^1, the 15th best individual receives 0.95^{14} and so on. This means, that we on average receive approx. five copies of the best fit individual, approx. one copy of the 33nd most fit individual, that the 46th individual survives with a probability of 50 % ,and that the least fit individual survives with a probability of 0.031 %. Fitness ranking conquers two of the biggest problems inherited from traditional fitness scaling: Over Compression and Under Expansion.

Over Compression occurs, in the beginning of a run, if a "super" individual with, say, ten times the average fitness, is created. As a consequence of the fitness scaling's attempt to avoid Premature Convergence, (the geometrical equivalence is being stuck at a local maximum in the 121-dimensional search space,) the different individuals fitness values will be far too close to each other. We then risk not only too slow convergence but also Genetic Drift away from the true global optimum (optima).

Under expansion is essentially another name for the same phenomenon, when it occurs at the end of a run. The genotypes are now supposedly very close to each other in fitness, and the fitness scaling will try to increase the distance among them, to avoid Slow Finishing. An extremely poor genotype, created perhaps by a mutation, will represent one limit to the expansion (and the best fit individual the other), and we will never quite reach the top of the function we are optimizing.

Fitness ranking represents an elegant solution to both these problems, as it always forces a user-defined function (in my case an exponential function) onto the individuals. No matter how good a solution, it will, on average, never survive more than five times each generation. And most reasonably fit individuals have a

reasonable probability of survival. Experimental evidence has by now shown fitness ranking to be superior to ordinary scaling over a wide spectrum of problems.

As selection technique I have chosen Stochastic Universal Sampling (SUS), invented by James Edward Baker. Baker shows [Baker 87], that SUS has Minimum "Spread" and zero "Bias". Bias has (in this case) nothing to do with the neurons' thresholds, but is an expression for the absolute difference between the expected and the actual number of individuals. Spread is here the set of possible numbers of individuals, that can be selected, by a given sampling strategy; minimum spread is defined as the theoretically smallest spread that allows zero bias. Baker thus effectively minimizes the genetic drift caused by selection. This makes SUS superior to for example Stochastic Remainder Sampling Without Replacement, a technique without the above-mentioned properties, that is often used as a kind of "standard" sampling technique in GAs. SUS can be visualized as a "wheel of fortune" with n arrows instead of one, with equal angle distance to each other, n being the number of individuals in the population.

I use a generation gap of one. When 100 individuals have been selected by SUS, these are paired off randomly, and the 50 pairs are crossed over. Both children of a crossing are mutated and passed on to the next generation, thus again constituted of 100 individuals.

2.4 Genetic Operators

I only use the two operators "crossover" and "mutation". In section 4.2 (Future Work) I discuss the possible appliance of some kind of reordering operator.

2.4.1 Crossover

I use a combination of the two techniques "Two Point Crossover" and "Uniform Crossover", which I have called "Two Point Uniform

Crossover with Constraints", which is specially designed to this problem.

Two-point crossover [Booker87] regards the chromosome as a ring instead of a string; i.e. two crossover points are chosen. Both experimental results and theoretical evidence, (one gets more building blocks, as these can be wrapped around the ends of the string. Schemata, which in the ordinary string representation, have the longest Defining Length, are now not so easily ruined by crossover.) lead to the conclusion, that two-point crossover is superior to ordinary crossover.

Uniform crossover is a more controversial matter, and many arguments exist, both experimental and theoretical, for and against it. See e.g. [Eshelman et al.89] and [Syswerda89]. But then, it is not really uniform crossover I use; "-with Constraints" refers to the fact, that I only allow crossover between the parts of two chromosomes, that represent the "same" weights. "Same" means, in this connection, bit sequences that are decoded to the same weight in the perceptron, i.e. the same position in the perceptron, *not* (necessarily) the same functionality. The chromosomes can thus, at the time of crossover, be regarded as consisting of a string of 121 rings, each made up of 53 bits. The two crossover points, for a given ring chosen for crossover, are selected at random. My justification of these constraints is the (reasonable) assumption, that a stronger epistasis exists between bits in the same weight, than between bits in different weights, statistically speaking.

It should be remarked, that my crossover technique in itself is a solution to the multiple representations problem, as it is unlikely, that a whole cluster of weights will be switched simultaneously.

The crossover rate mentioned in section 2.5 means the probability that each single weight is crossed. Finally it should be mentioned that a crossover is only performed, if there is a distance of at least two bits between the crossover sites. Otherwise it is canceled.

2.4.2 Mutation

I use ordinary bit mutation where each bit in the chromosome is flipped with a certain probability. This is *not* the same as each bit acquiring a random value with the above-mentioned probability, but, on average, with twice the above mentioned-probability. This banal point often causes misunderstanding in the literature.

2.5 Parameter Values

By way of some short, initial experiments I fixed the crossover rate at 0.01 and the mutation rate at 0.001, which means, that on average one weight of 100 will be crossed and one weight of 19 will be mutated each generation. This again corresponds to each chromosome being crossed 1.2 places and mutated in 6.4 bits on average. With a higher mutation rate, the system performed almost like a random search, as the mutations override the selection mechanism. This is characterized by the fact that a large part of the population, as mentioned in section 2.2, either rejects or accepts all grains. In other words, the GA does not function as a stochastic hillclimber. If no mutation is used, the population converges to a local maximum in very few generations.

The population size is 100. This is probably a bit on the high side of what one would usually choose, but I think that the overhead in computer time is worth the (relative) security, a large gene pool provides. (As a matter of fact, many of the results in GA theory are only valid, under the assumption, that the population size is infinite!)

The number of generations is 120, which is based on the experimental observation that almost all runs will have converged at this point.

The above choices are hardly optimal, and there are other ways to determine a GA's parameters. One possibility is a theoretical calculation of the crossover and the mutation rates. Such techniques

have been developed by, for example, DeJong [DeJong75]. But as the GA I use differ from a standard GA (on, among other things, the crossover operator), non of the methods known to the author, can be directly applied here. Another possibility is to use a Meta GA to find near optimal parameter settings [Grefenstette86]. The idea is theoretically attractive and has in practice yielded positive results. But it is rather computationally heavy. Both the above-mentioned methods are outside the scope of this article, and one can find comfort in the thought, that GAs are supposed to be rather parameter stable.

3 Results

The results of my experiments follow. In each section the relevant characteristics of the experiment are described. The graphs show the best of ten runs, where "best run" is defined as the run that obtained the best single individual. Both the average and the best individual of the generation are shown.

3.1 Standard Configuration (Reference)

Here the raw fitness is calculated from a training set consisting of 1000 wanted and 990 unwanted grains; i.e. approx. the same quantity of the two categories. Each correctly classified grain counts 1.0 in raw fitness, i.e. a raw fitness of max. 1990 is possible. When a potentially successful solution is obtained, the test set, consisting of 2000 wanted and 4500 unwanted grains, is propagated. The perceptrons generalization ability is thus thoroughly tested. As mentioned in section 1.1, a successful solution is defined by

"No. of correctly classified "wanted" grains" greater or equal to 85.0 % of all wanted grains

"No. of correctly classified "unwanted" grains" greater or equal to 99.0 % of all unwanted grains.

The results are shown in fig. 5. However, with a single modification: The fitness ought to be reproduced in two dimensions as a function of the number of generations. One dimension for the wanted grains and one for the unwanted grains. See fig. 4. As 3D graphs seldomly turn out well on paper, I have instead chosen, in the depiction, to add 150.0 to the raw fitness of potentially successful solutions. The lowest value, which ensures that the best solutions will in fact have the highest fitness. The maximum theoretically possible fitness then becomes 1000.0 + 990.0 + 150.0 = 2140.0. It is emphasized, that this boost of the raw fitness is *not*

used during the GA's run. It is solely a technical "trick" in the depiction.

FIGURE 4 HERE

As appears from fig. 5, a population of reasonably fit individuals, quickly arises and appears to remain reasonably stable. Five individuals, which potentially fulfill the minimum requirements are

FIGURE 5 HERE

created. Four of these, are able to generalize satisfactorily. The best individual is created in generation 111, with "No. of correctly classified "wanted" grains" (WG) equal to 85.85 % and "No. of correctly classified "unwanted" grains" (UG) equal to 99.13 %, in performance on the test set.

The reason for the relatively modest number of successful solutions is of course, that the GA has another goal in sight than the one we have defined. Its only interest is to optimize its raw fitness, and therefore weights the recognition of wanted grains just as highly as the rejection of unwanted grains. From our viewpoint, the successful solutions are just spin-offs, from a less discriminating search for a generally good solution (in this run, the record is WG=98.00% and UG=96.26%). In the next experiment I will try to change the situation, by the introduction of Penalty Terms.

3.2 Penalty Terms

We need a technique to guide the GA's blind search in the right direction, which traditionally can be obtained in two ways: We can either use "Enforcement" or "Penalty Terms". The former strategy bans chromosomes, that which do not measure up to the minimum

requirements; i.e. they are “killed off” without any possibility of contributing genetically to the next generation. The remaining strings, above the minimum limit, then somehow fill up the empty space in the gene pool. But this technique demands, that our minimum requirements are less severe than the case being, as it would be difficult to fill up the initial population. (One would need to collect fit chromosomes through many runs like the one in section 3.1.)

The latter strategy, the one I will use in this section, consists of punishing (or rewarding!) an individual’s raw fitness, before the fitness ranking is performed, according to how close it is to the optimal requirements. The following lines of pseudo code, equal the penalty terms placed in the C program:

```
IF (#WG >= 85.0% AND #UG >= 98.0%) THEN
     raw_fitness = raw_fitness + 120.0
     raw_fitness = raw_fitness + 6.0*(#UG - 990.0*99.0%)
END IF

IF (#WG >= 87.0 %) THEN
     raw_fitness = raw_fitness - 2.0*(#WG - 1000.0*85.0%)
END IF
```

The first term, and its consequences, are immediately intelligible. (The numeric constants is a qualified guess from the author). An individual, who (almost) lives up to the minimum requirements, is rewarded. The reward consists of a higher fitness, which directly leads to bigger influence on the next generation’s genes. The last three lines of the pseudo code are less obvious. Often, while training the perceptron by means of BP, it was remarked that, in very good solutions, there was a trade-off between WG and UG; when one of them went up, the other went down. Therefore I punish successful solutions, that use too many “resources”, (meaning internal flexibility in the neural net) on increasing WG, (supposedly) at UG’s expense. When WG is just over 85 % we are satisfied with it, and the attention must be focused on optimizing UG.

The results of the best of ten runs are shown in fig. 6, the maximum theoretical possible fitness here being (besides extremely unlikely) 1000.0·0.87 + 990.0·1.0 + 120.0 + 6.0·(990.0·1.0 -990.0·0.99) = 2039.4.

FIGURE 6 HERE

Note that, while the average fitness after about 50 generations stabilizes at an average value, with oscillations in the size of 5-10 %, the fitness of the generation's best individual continues to grow more or less all through the experiment. And we do end up with no less than 2093 potential solutions, and 1053, i.e. more than half of these, generalize satisfactorily on the test set! It is worth mentioning, that in the last generations, there are more occurrences of potentially good solutions that turn out to generalize worse, than the potential solutions in the earlier generations, while at the same time, the potential solutions that DO generalize satisfactorily, are better than the ones in the previous generations. This polarization is caused by the fact, that some of the chromosomes have stored noise contained in the training set; a phenomenon well known in NN circles, where it is named overfitting (or over-training). This indicates that the networks, to a certain degree, lack more training data- they are in theory capable of storing more information/rules, and have indeed done this, unfortunately in the shape of noise. I will return to this important point in the next section. The best net in this experiment has WG=85.70% and UG=99.20%; the best so far. This phenotype occurs in generation 103.

3.3 Composition of the Training Set

Now I alter the composition of the training set. The one we have used so far was composed of 1000 wanted (i.e. wheat) grains and 990 unwanted (i.e. rye, barley and oat) grains. I try to compensate

for the lack of training data, mentioned in the last section, by changing this relationship. As we want the highest precision in connection with UG, it makes sense to give the GA access to more instances of unwanted grains, so it can build up a statistically more stable, a more coherent and thus better, "picture" of the fitness value as a function of the 121 weights in the perceptron. A quasi-logic choice is to let the training set contain 15 times as many unwanted grains as wanted grains, as we want a 15 times higher precision (1.0%) with UG than with WG (15.0%). In the following run (best of ten), all conditions are equal to the ones in section 3.1,

FIGURE 7 HERE

apart from the fact that the training set now consists of 128 wanted and 1854 unwanted grains. The test set is unaltered (2000 wanted, 4500 unwanted). The maximum theoretical possible fitness is now 128.0 + 1854.0 = 1982. The results are presented in fig. 7.

Again the average individual is seen to oscillate around a rather fixed value, while the generation's best individual's fitness is increasing as a function of the number of generations, through the whole run. This time there is "only" 232 potential solutions. As the test set is propagated, 47 of these turn out to generalize satisfactorily. The best phenotype is present both in generation 119 and generation 120. (One gets the impression, that 120 generations was a bit on the low side for this specific run). It has WG=85.40% and UG=99.36%. We have now, with a factor ten fewer successful solutions, broken the record from the previous experiment. It would seem, that we have increased the exploitation/exploration ratio. It would now be natural to try to combine the methods from section 3.2 and this section. I will return to this in section 3.7.

3.4 Real Value Raw Fitness

In this section I will return to the "roots", as I replace the discrete error function I have used so far, with a continuous one, similar to the one most often used during traditional BP training of perceptrons. The hitherto used error function has been discrete in the sense that a positive output, caused by the propagation of a given input pattern, has been read as a +1.0, while a negative output has been regarded as a 0.0, and then added to the fitness score. Nobody has actually cared if the answer was e.g. 0.4 or 0.9. In this experiment we use the classic quadratic error measure $\left(\zeta_i^\mu - O_i^\mu\right)^2$, where ζ_i^μ is the expected (wished for!) output, and O_i^μ is the actual output. This number is summarized for the whole training set and thus constitutes the raw fitness function. It is an obvious increase of the exploitation/exploration ratio, as the error landscape now no longer contains "don't care" plateaus. The hope is now that the GA in a more unambiguous way will be able to find a road leading to the existing global minimum (or minima). (With this new fitness function we are looking for minima, not maxima). Otherwise, the conditions are similar to the conditions in section 3.2, apart from the constants on the penalty terms which have been adjusted to fit the magnitudes of the new, continuous, quadratic fitness function.

Out of 10 runs (or, as a matter of fact 20, I performed 10 more, to be sure), not one potential solution that turns out to generalize satisfactorily occurs. The best run of the first 10 has no less than 4006 potential solutions, but non of them reached the 85.0/99.0 limit in performance on the test set! The GA obviously lacks the freedom to act it is given by the discrete fitness function. The bad generalization ability would suggest that we have reached "noise-fissures" in the energy landscape of the training set. Maybe the GA enters these, because the new, cogent fitness function forces genotypes which want a chance of survival downwards at any price.

So why is the quadratic error measure generally working well (if not optimal) when BP is used in perceptron training? One explanation which springs to mind is that the stringent error measure is countered by the randomness inherent in online BP. The slightly different versions of the error landscape contained in each input instance makes the error landscape more "shimmering" and it is thus harder to get stuck in a noise-fissure. GA training is more reminiscent of batch BP training.

But the main reason for the relative success of the quadratic error measure in BP is of course, that BP is a local search procedure, a gradient descent algorithm which "makes the best of a bad job"; i.e. it descents whenever possible, and hopes to get out of local minimas by noise. In such a situation the quadratic error measure is a reasonable measure of success. It makes little or no sense however, to transfer this measure of success to solutions produced by a GA's holistic (i.e. produced primarily by crossover) hyperplane sampling. The gradient information contained in the quadratic error measure is of no immediate use to the GA. As we do not know the geometric expressions of the GA's *implicit parallelism* [Goldberg 89a], the only *objective* measure of success is how well the perceptron performs the classifications. So to avoid imposing unreasonable constraints (which might be downright harmful) on the GA's search of the perceptron's error landscape, I simply use the number of correct classifications (maybe weighted and maybe with some penalty terms as well) as the basis of a fitness ranking. As is evident from the above, this approach in its apparent simplicity completely outperforms the quadratic error measure. Notice that, unlike the quadratic error measure, my fitness function is completely *monotonic* [Hampshire 92]; a quality not easily obtained for error functions used for BP. See [Moeller & Depenau94] for a discussion of ways to approach monotonicity during BP.

Goldberg has recently pointed out the need for defining landscape characteristics important to crossover, unless we want to continue to look at the fitness-landscape from a hillclimber's (i.e. gradient ascent) point of view [Goldberg & Horn 94].

It is worth to emphasize that during GA training of neural nets, the quadratic fitness function is always used, by direct inspiration from back-propagation. Indeed it is recommended as a standard in a survey paper of the GA/NN field by David Schaffer [Schaffer et al.92]. It is more than likely that this is the main reasons for GAs' undeserved bad reputation in this area. Examples such as the above show that genetic algorithms, though being a general search technique, should not be applied to a problem "mindlessly". They must be carefully fitted to the concrete problem if one hopes to achieve good results.

3.5 Hillclimbing of Good Solutions

Now we try to hillclimb all solutions that generalize satisfactorily on the test set. I therefore use three different data sets (otherwise some spoil-sport might cry foul): One training set, for the fitness function, as previously. One test set, to decide whether to hillclimb a given solution, or not, and a validation set which takes on the role the test set had in the former sections, to test the final generalization ability of the phenotypes. Like the test set, the validation set consists of 2000 wanted grains and 4500 unwanted grains. The hillclimber is a non-stochastic, round robin climber that in turn flips the $29 \cdot 121 = 3509$ bits in the genotype, which code for the 30 most significant bits in all 121 weights in a given phenotype. To keep down the number of candidates, the prerequisite to be hillclimbed is a performance with WG>85.0% and UG>99.1% on the test set.

For each inverted bit, the chromosome is decoded and the training set is propagated. If the performance is increased (defined

as UG increased, and WG not below 85.0%), the change is kept, otherwise the bit is flipped back to its original value. The climber runs, on a given chromosome, till it has not succeeded in improving the performance 3509 times in succession; i.e. it runs at least 3509 times and at most an infinite number of times (or to be precise $(3509 - 1) \cdot 2^{3509}$ times). In the end the validation set is propagated. It must be stressed that the round robin hillclimber has nothing to do with Whitley's "Genetic Hillclimber". The former is a superstructure often imposed on GAs. Geometrically speaking, it means that one will always "at least" reach a local extremum, in the fitness landscape. (It appears silly to stand half way up a mountain side and apparently have no idea of how to proceed further). Apart from the hillclimber, the conditions in this experiment are like in section 3.2. Of the runs I started, many did never finish, due to the hillclimber's great need of processing power.

The hillclimber turned out not to be worth the effort. During the execution of one hillclimbing cycle (requiring 3509 decodings of the chromosome, and 3509 propagations of the training set), an amount of processing power equal to almost 35 standard generations' needs was used ("standard" meaning a generation not containing any successful individuals; a standard generation takes about 60 sec. in real time to perform on an HP 705), and in most cases the acquired advantages turned, out to be minimal. Most often, no progress at all is made in any of the 121 dimensions! And almost never (one in a hundred) in more than two or three. For a GA advocate, it is quite satisfying, that the GA's blind hyperplane sampling finds local maxima in a 121-dimensional space. (Or to be more precise, a 121 times 53, i.e. 3509-dimensional binary space). Evidently from a more practical point of view, it means that the hillclimber was a dead end.

Fig. 8 shows the hillclimb that had the most success, concerning improvement of the already acquired weights. It is, with WG=85.20% and UG=99.29% also the record in this section.

Each “hop” corresponds to one bit being tested. In more than two cycles, i.e. during more than 7018 propagations, there are five bits, the flipping of which produces an improvement on the performance. The screen dump also illustrates the above-mentioned trade-off between WG and UG; when one goes up, the other goes down.

FIGURE 8 HERE

3.6 Alternative Initialization

Instead of creating the initial generation of perceptrons, as described in section 2.2, I now use a completely uniform random initialization. Each bit in the initial population has exactly a 50 % probability of being switched on. This is the procedure most often recommended for the seeding of a GA, with the argument that then there will be amble instances of each allele at all loci. Then it is up to the genetic operators to sort and recombine these. In practice this turns out *not* to hold good. I performed 10 runs, like the ones in section 3.2 with this initialization. The best result was WG=86.20% and UG=99.06%. The last generation turned out, in all ten cases, to contain very alike phenotypes. That could point to some kind of premature convergence. The bad results are probably produced by the fact that many of the initial weights, produced by the new initialization strategy, lie in the interval where the perceptron’s activation function is completely saturated. (The activation function

is the usual hyperbolic tangent function). The same conclusion as in section 3.4 can be drawn: The GA must be adjusted to the specific problem in question.

3.7 Last Experiment

As a last experiment I try to combine the best of the previous experiments. I use constraints and have increased the training set considerably to 1344 wanted and 4050 unwanted grains; The maximum number of training data available. Besides I have increased the number of generations to 200. Finally, the hillclimber will be activated if a successful solution has UG>99.30%. Ten runs gave as the best result WG=85.20% and UG=99.31% and the hillclimber did not improve this solution. An interesting phenomenon occurred in almost all ten runs: Rather few potential solutions, typically around 40 in a run, were found. But they had a very fine generalization ability, when the test set was propagated; often with UG in the area of 99.20% to 99.28%. This situation did not occur during the experiments described in section 3.1 to 3.3. I have no immediate explanation for this.

4 Conclusion and Future Work

4.1 Conclusion

I have in this paper tried to argue that it is possible to train perceptrons efficiently by way of genetic algorithms, if one is careful to observe that a series of problem-specific points, such as initialization, chromosomal representation, crossover technique and, most importantly, choice of fitness function are non-trivial, and deserve special attention.

The strategy for applying my fitness function to a general problem is the following: First Count the number of correct classifications (or whatever) the phenotype makes on the training

set. (In situations where we have multiple output nodes, we simply count, maybe weighted, the number of output nodes which answer correctly, for each training pattern, and then sum up for all the patterns. The absolute size of a phenotype's raw fitness do not matter as it is only compared to other phenotypes, which have had their raw fitness calculated in the same way, during the fitness ranking). Then we might lay some problem specific penalty terms on the raw fitness as in section 3.2. Finally we perform a fitness ranking, and acquire the reproductive fitnesses used for the selection. I have used an exponential ranking, but a linear ranking might be used with just as much success. Of course one might also try to use some gradient information, to choose between two perceptrons which have the same performance in terms of pure classifications. Here it might be sensible to choose the perceptron whose answers lie closest to the output neurons' activation functions' saturation, as this perceptron might have more internal flexibility left, than a perceptron which barely manages to get the correct sign from the activation function. Such an approach could build on a raw fitness consisting of the number of correct classifications plus a gradient term. For inspiration see [Moeller & Depenau94].

It is difficult to directly compare GA training with the various kinds of back-propagation. The GA has a larger space requirement than back-propagation, as a certain population size must be sustained. But the space requirements should not be a problem on most machines today, which are usually equipped with at least a couple of megabyte RAM in their standard configuration. More relevant is the time requirements, i.e. the processing power required. It does not make a lot of sense to talk about time complexity in this connection since, as has often been pointed out, the time used to adjust the many loose parameters, that both genetic algorithms (population size, probabilities of applying genetic

operators, generation-gap size etc.) and back-propagation (learning-rate, momentum, error measures etc.) suffer from, should be included objectively in the final time measure. Maybe there is something to be said for GAs here, as they are known to be relatively parameter insensitive, and thus easier to steer to an acceptable configuration. BP training, on the other hand, probably has fewer loose parameters than GAs. It should be mentioned that GAs are very good at creating a large number of different, good solutions to a given problem. And these may be further trained by various techniques. In a recent, comparative study of NN learning-techniques [Heistermann94], GA training with a subsequent gradient search optimization yields good results.

Concerning results, I achieved a 99.36% recognition of bad seeds on the test set. In the autumn of 1993, 99.44% was reached by way of back-propagation, a little better than the former. On the other hand 99.44 % was the best result achieved by approx. 40 people, so the results I have obtained cannot be overlooked. And the generalization abilities of the GA networks are generally very high.

It should also be noticed that the training techniques presented in this paper is in no way restricted to perceptrons only. It can be used on any network architecture, including recurrent architectures which are not easily trained by traditional methods. As a matter of fact, their is reason to believe that “pure” perceptrons, due to their many symmetries, represent a kind of worst-case architecture.

Furthermore, there is reason to believe that GA training has a larger potential than has been investigated here. This is what the next section is about.

4.2 Future Work

Due to the many loose parameters, mentioned in the preceding section, and due to the many possible alternatives under each point in Davis & Steenstrup’s five component outline [Davis87], there is

ample opportunity for future work. But there is in particular one possibility that should be mentioned, as it can be expected to yield good results. It is connected to the multiple representations problem, see fig. 9. I have, as mentioned in section 2.1, to a certain

FIGURE 9 HERE

degree tried to solve this problem by coding the perceptron in such a way that a node in the hidden layer has its incoming weights distributed across most of the chromosome; see fig. 3. In this way, the GA is to a certain degree forced to more or less "choose" one of the possible permutations early in a run, as a templet for future generations to work on. But a better solution to the problem would be some kind of reordering operator. An effective reordering operator would completely dispose of the problem. Unfortunately, reordering is a highly non-trivial problem, (as a matter of fact, much harder than the original problem of finding good values, i.e. weights) and the operator usually employed for the purpose, Inversion, is notorious for its lack of efficiency. Baker has given some theoretical arguments for the lack of success of inversion. With this in mind, Goldberg has, some time ago [Goldberg et al. 89b] suggested a new kind of GA, a "Messy Genetic Algorithm" (mGA):

A mGA contains chromosomes of variable lengths. This means that the phenotype (in our case the perceptron) can be over- or underspecified. The latter requires that one is able to define a partial fitness function based on a local maximum in the search space. (One can see a possible rehabilitation of the round robin hillclimber here). This is without doubt the greatest difficulty in applying a mGA, but in our case it should be possible to solve it. In this connection the interesting thing about Goldberg's invention is the

operators. The mGA uses mutation just as an "old-fashioned" GA, but instead of crossover (and inversion) it has two operators called Cut and Splice. As their names more than suggest, the former divides a chromosome in two, while the latter splices two chromosomes together. Goldberg argues convincingly, from theoretical arguments, that the cut/splice operators not only work effectively on the chromosome's alleles (like crossover), but also work on their ordering (i.e. their coding) much more effectively than inversion in an ordinary GA. Furthermore he tests the mGA, with success, on a 30 bits order-three-deceptive function with 3^{10} optima. A situation not unlike ours, in character if not in size. It is easy to imagine that this new technique should be able to address "The Multiple Representations Problem" somehow, and the mGA thus appears to be tailor-made for the training of perceptrons. More information about "messy genetic algorithms" can be acquired from [Goldberg et al.89b], [Goldberg et al.90], Goldberg et al.91], [Merkle and Lamont93] and [Markus93].

5 Acknowledgments

I would like to thank Prof. Brian Mayoh and Jan Depenau at DAIMI and Prof. David Goldberg at the University of Illinois for helpful suggestions. And a big thank you to Ralf Mattethat at DAIMI for critisism, for struggling with the word processor and for moral support.

References

Ackley, D.H. (1987). An Empirical Study of Bit Vector Function Optimization. Genetic Algorithms and Simulated Annealing, pp. 170-204. Morgan Kaufmann.

Baker, J.E. (1985). Adaptive Selection Methods for Genetic Algorithms. Proceedings of an International Conference on Genetic Algorithms and their Applications, pp. 101-111.

Baker, J.E. (1987). Reducing Bias and Inefficiency in the Selection Algorithm. Proceedings of the Second International Conference on Genetic Algorithms, pp. 14-21. Lawrence Erlbaum Associates, Publishers.

Booker, L. (1987). Improving Search in Genetic Algorithms. Genetic Algorithms and Simulated Annealing, pp. 61-73. Morgan Kaufmann.

Caudell, T.P. and Dolan, C.P. (1989). Parametric Connectivity: Training of Constrained Networks using Genetic Algorithms. Proceedings of the Third International Conference on Genetic Algorithms, pp. 370-374. Morgan Kaufmann.

Davis. L. and Steenstrup, M. (1987). Genetic Algorithms and Simulated Annealing: An Overview. Genetic Algorithms and Simulated Annealing, pp. 1-11. Morgan Kaufmann.

Davis, L. (1989). Adapting Operator Probabilities in Genetic Algorithms. Proceedings of the Third International Conference on Genetic Algorithms, pp. 61-69. Morgan Kaufmann.

DeJong, K.A. (1975). An Analysis of the Behaviour of a Class of Genetic Adaptive Systems. University of Michigan Press.

Eshelman, L.J., Caruna, R. and Schaffer, J.D. (1989). Biases in the Crossover Landscape. Proceedings of the Third International Conference on Genetic Algorithms, pp. 10-19. Morgan Kaufmann.

Fahlman, S.E. (1988). Fast-Learning Variations on Back-Propagation: An Empirical Study. Proceedings of the 1988 Connectionist Models Summer School. pp. 38-51. Morgan Kaufmann.

Fahlman, S.E. and Lebiere, C. (1990). The Cascade-Correlation Learning Architecture. Advances in Neural Information Processing Systems II, pp. 524-532. Morgan Kaufmann.

Fahlman, S.E. (1991). The Recurrent Cascade-Correlation Architecture. CMU-CS-91-100, School of Computer Science, Carneige Mellon University, Pittsburgh.

Goldberg, D.E. (1989a). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley.

Goldberg, D.E., Korb, B. and Deb, K. (1989b). Messy Genetic Algorithms: Motivation, Analysis, and First Results. Complex Systems 3, pp. 493-530.

Goldberg, D.E., Korb, B. and Deb, K. (1990). Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale. Complex Systems 4, pp. 415-444.

Goldberg, D.E., Korb, B. and Deb, K. (1991). Don't Worry, Be Messy. Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 24-30. Morgan Kaufmann.

Goldberg, D.E. and Horn, J. (1994). Genetic Algorithm Difficulty and the Modality of Fitness Landscapes. FOGA 3, DRAFT 8/15/94.

Grefenstette, J.J. (1986). Optimization of Control Parameters for Genetic Algorithms. IEEE Transactions on Systems, Man and Cybernetics.

Hampshire, J.B. (1992). A Differential Theory of Learning for Statistical Pattern Recognition with Connectionist Models. Ph.D. Thesis, School of Computer Science, Carnegie Mellon University.

Heistermann, J. (1994). Different Leaning Algorithms for Neural Networks - A Comparative Study. The Third Conference on Parallel Problem Solving from Nature. pp. 386-396. Springer-Verlag.

Hertz, J., Krogh, A. and Palmer, R.G. (1991). Introduction to the Theory of Neural Computation. Addison-Wesley.

Holland, J.H. (1975). Adaption in Natural and Artificial Systems. University of Michigan Press.

Korning, P.G. (1994). Training of Neural Networks by means of Genetic Algorithms Working on Very Long Chromosomes, technical paper-DAIMI PB-486. University of Aarhus Press.

Markus, A. (1993). Dual Insights into Genetic Algorithms. Proceedings of the Fifth International Conference on Genetic Algorithms, P. 645. Morgan Kaufmann.

Merkle, L.D. and Lamont, G.B. (1993). Comparison of Parallel Messy Genetic Algorithm Data Distribution Strategies. Proceedings of the fifth International Conference on Genetic Algorithms, pp. 191-198. Morgan Kaufmann.

Miller, G.F., Todd, P.M. and Hedge, S.U. (1989). Designing Neural Networks using Genetic Algorithms. Proceedings of the Third International Conference on Genetic Algorithms, pp. 379-384. Morgan Kaufmann.

Montana, D.J. (1991). Automated Parameter Tuning for Interpretation of Synthetic Images. Handbook of Genetic Algorithms, pp. 282-311. Van Nostrand Reinhold.

Moeller, M. and Depenau, J. (1994). Soft-Monotonic Error Functions.

Pratchett, T. (1991). Reaper Man. Corgi Books, Transworld Publishers Ltd.

Romaniuk, S.G. (1993). Evolutionary Growth Perceptrons. Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 334-341. Morgan Kaufmann.

Ronald, E. and Schoenauer, M. (1994). Genetic Lander: An Experiment in Accurate Neuro-Genetic Control. The Third Conference on Parallel Problem Solving from Nature. pp. 452-461. Springer-Verlag.

Solla, S.A., Levin, E. and Fleisher, M. (1988). Accelerated Learning in Layered Neural Networks. Complex Systems 2. pp. 625-639.

Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms. Proceedings of the Third International Conference on Genetic Algorithms, pp. 2-9. Morgan Kaufmann.

Whitley, D. (1989). The Genitor Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. Proceedings of the Third International Conference on Genetic Algorithms, pp. 116-121. Morgan Kaufmann.

Whitley, D., Dominic, S. and Das, R. (1991). Genetic Reinforcement Learning with Multilayer Neural Networks. Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 562-569. Morgan Kaufmann.

Zhang, B.-T. and Myhlenbein, H. (1993). Genetic Programming of Minimal Neural Nets Using Occam's Razor. Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 342-349. Morgan Kaufmann.

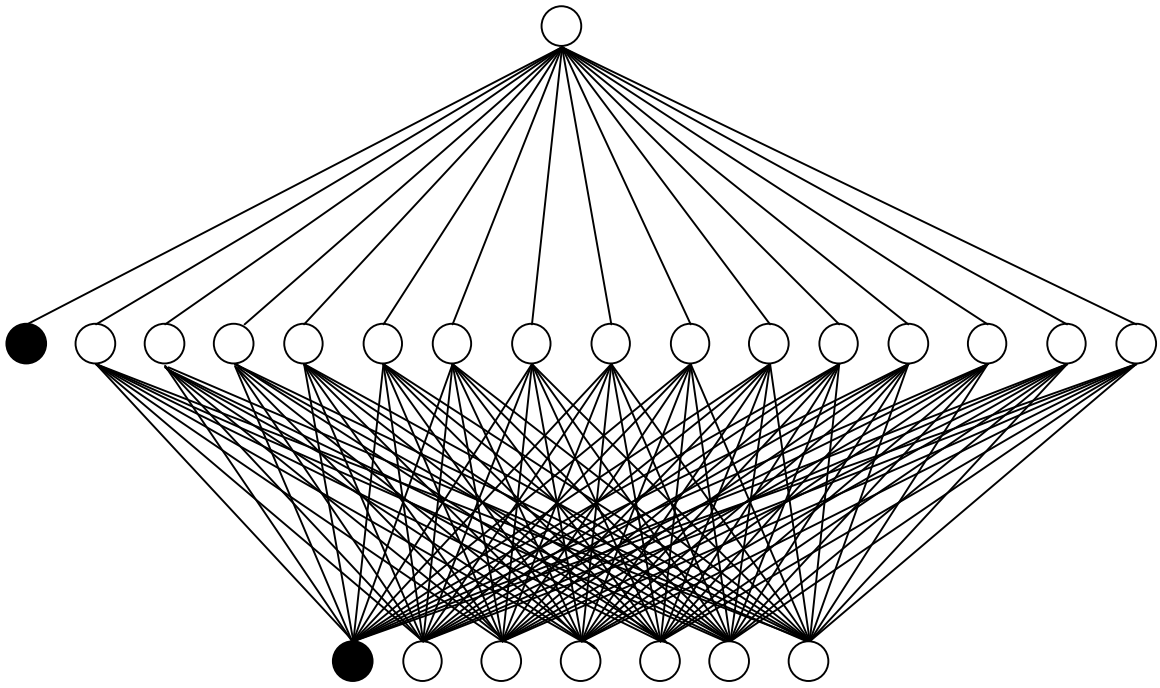

Figure 1. The Perceptron. The black nodes represent thresholds and are hardwired to the value -1.0.

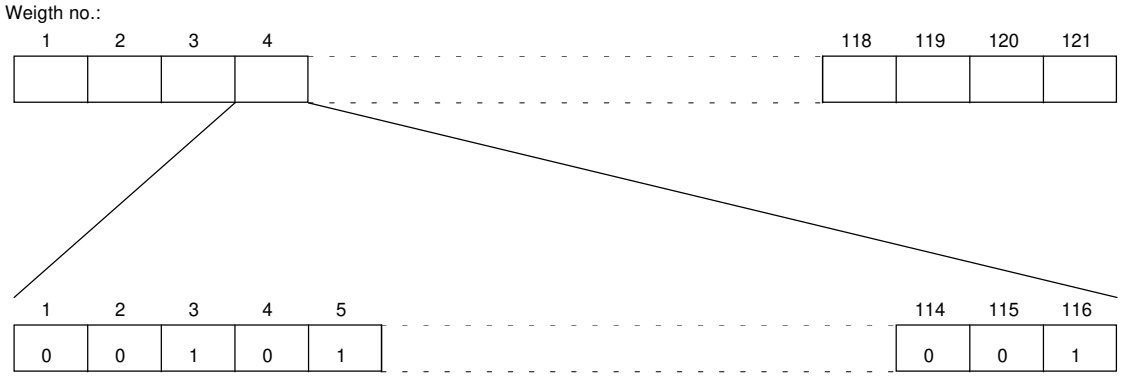

Figure 2. The Chromosome. Each of the 121 weights consists of 4 times 29 bits, i.e. 116 bits.

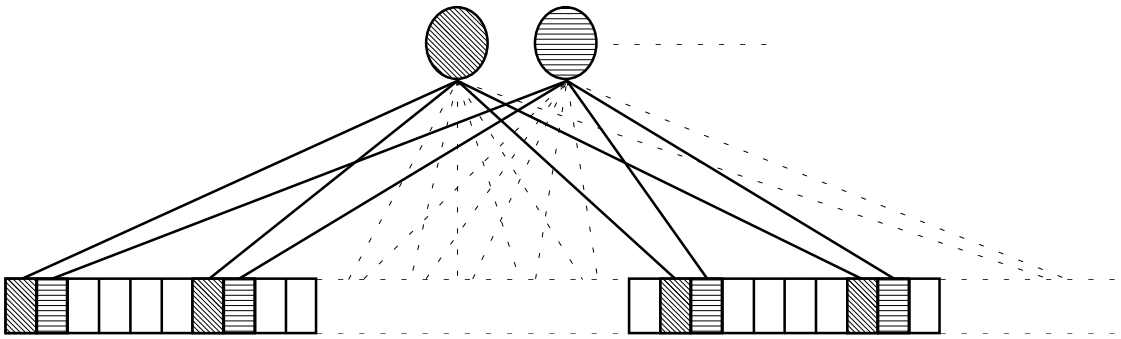

Figure 3. The bit groups, that each represent a weight, have been distributed over a large part of the chromosome. For the input/hidden-layer weights it means a concatenation of 7 clusters. Each cluster consists of a weight from each of the hidden neurons, i.e. 16 weights altogether .

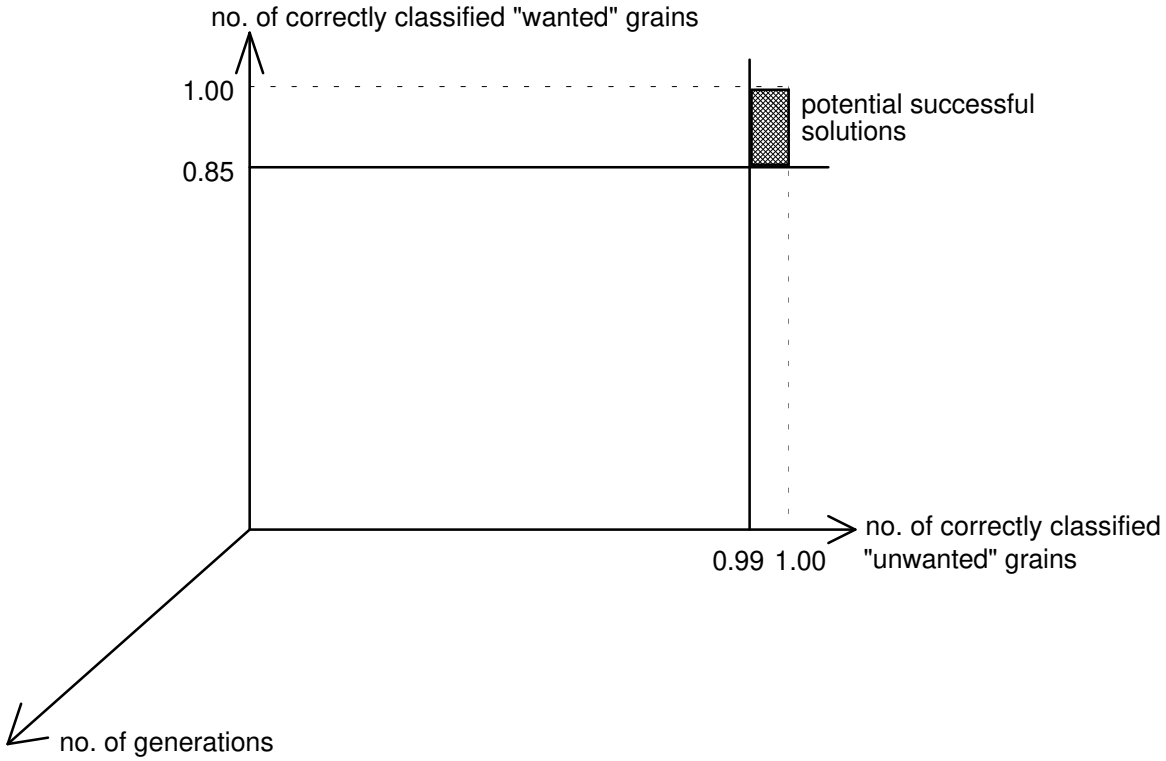
no. of correctly classified "wanted" grains
1.00
0.85
potential successful solutions
no. of correctly classified "unwanted" grains
0.99 1.00
no. of generations

Figure 4.

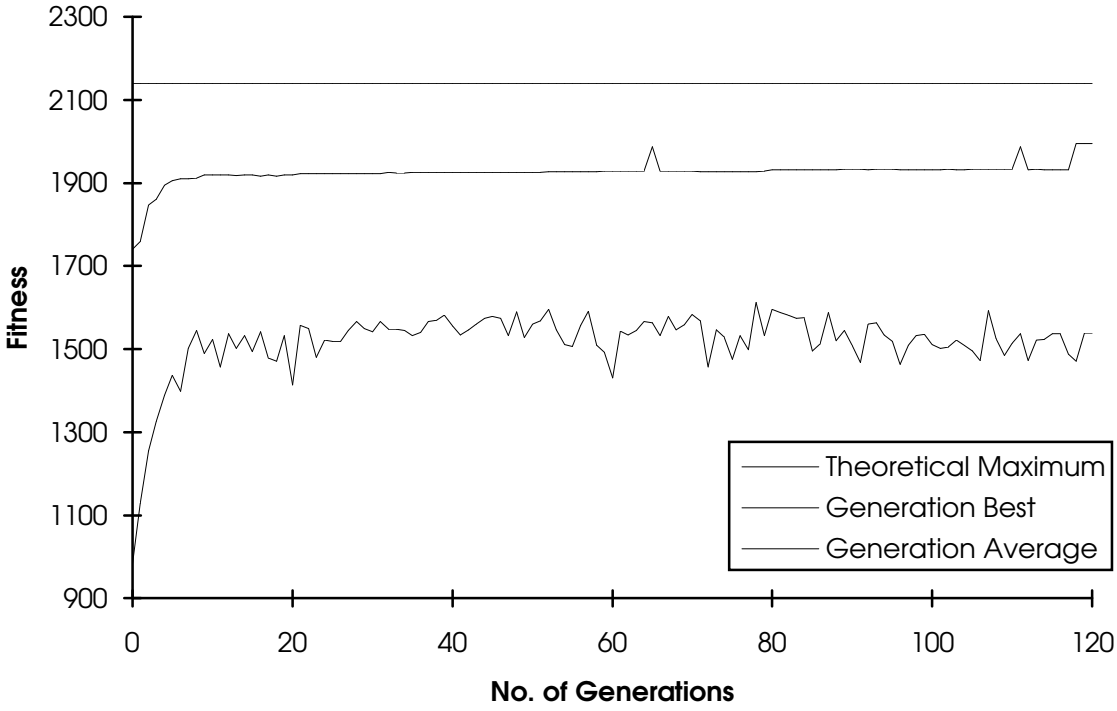

Figure 5. Standard Configuration (Reference).

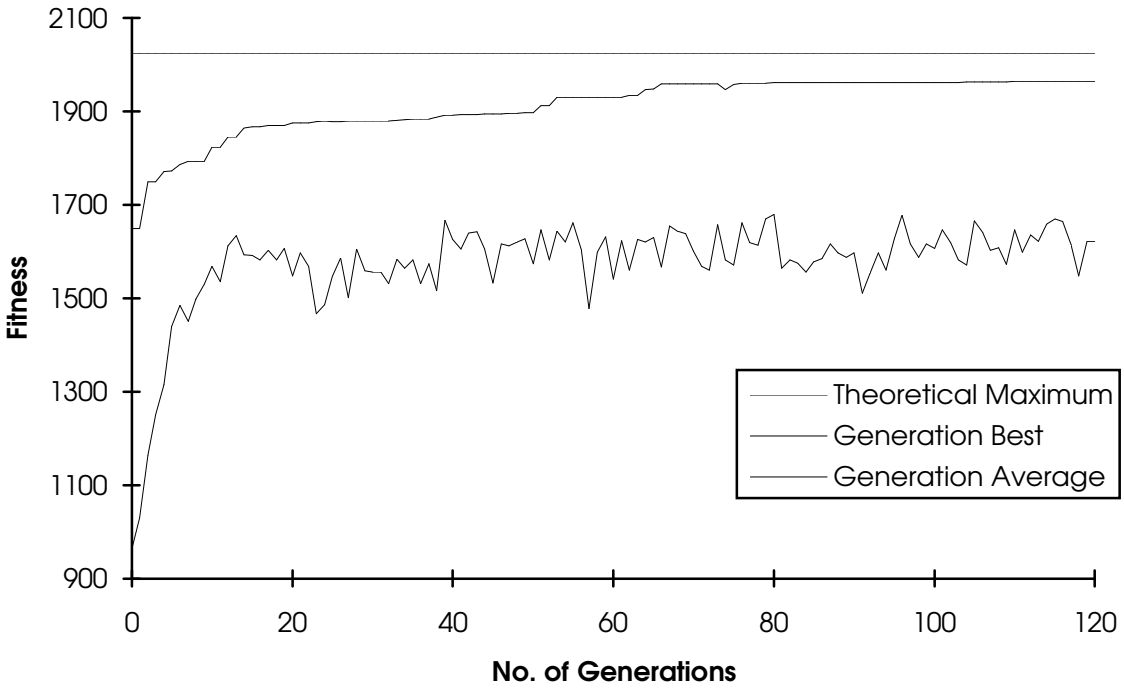

Figure 6. Penalty Terms.

Figure 7. Composition of the Training Set.

```
chro 46: WG:  97.023810 %  UG: 96.246914 %
chro 47: WG:  97.023810 %  UG: 96.271605 %
chro 48: WG:  78.720238 %  UG: 99.086420 %
chro 49: WG:  96.949405 %  UG: 93.777778 %
chro 50: WG:  85.416667 %  UG: 99.061728 %
ALERT! -- POTENTIAL SUCCESSFUL SOLUTION -- TEST-SET PROPAGATING....
CHRO 50: WG:  85.550000 %  UG: 99.111111 %

 in hop 161: an improvement on the TRAINING-SET
chro 50: WG:  85.193452 %  UG: 99.086420 %

 in hop 1361: an improvement on the TRAINING-SET
chro 50: WG:  85.267857 %  UG: 99.111111 %

 in hop 1822: an improvement on the TRAINING-SET
chro 50: WG:  85.119048 %  UG: 99.135802 %

 in hop 3131: an improvement on the TRAINING-SET
chro 50: WG:  85.119048 %  UG: 99.160494 %
CYCLE 1.000000 IS NOW COMPLETED

 in hop 5071: an improvement on the TRAINING-SET
chro 50: WG:  85.044643 %  UG: 99.185185 %
CYCLE 2.000000 IS NOW COMPLETED


ATTANCION!!! time to watch the (possible) improvement......
VALUATION-SET PROPAGATING
CHRO 50: WG:  85.200000 %  UG: 99.288889 %
END OF HILLCLIMB!!!!!!!!!!!!!!!!!!!!!

chro 51: WG:  99.925595 %  UG: 75.629630 %
chro 52: WG: 100.000000 %  UG: 11.901235 %
chro 53: WG:  96.502976 %  UG: 96.395062 %
chro 54: WG:  99.851190 %  UG: 86.592593 %
chro 55: WG: 100.000000 %  UG: 52.938272 %
chro 56: WG
```

Figure 8. Screen dump of the Hillclimber In Action.

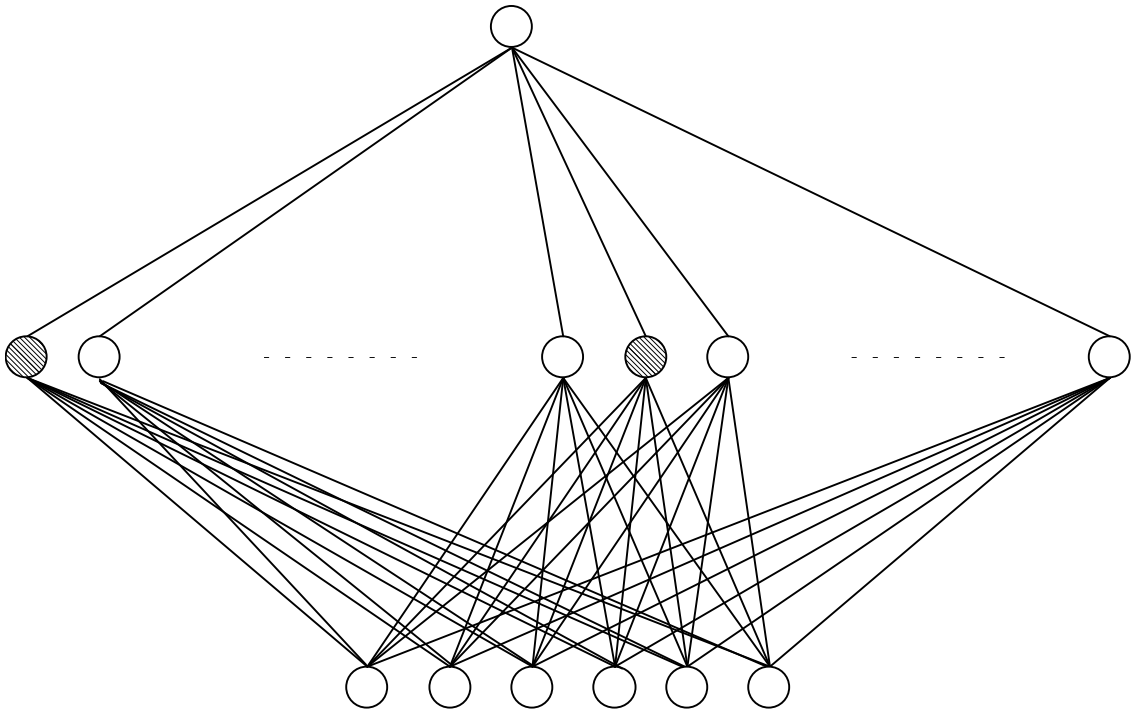

Figure 9. The hatched neurons (or rather, their weights) can switch positions without altering the perceptron's functionality.