# A Study On Hill Climbing Algorithms For Neural Network Training

**Stephan Chalup** **Frederic Maire**
Machine Learning Research Centre, Neurocomputing Laboratory
School of Computing Science, Faculty of Information Technology
Queensland University of Technology
2 George Street, Brisbane Qld 4001, Australia
phone: +61 (0)7 3864 2575, fax: +61 (0)7 3864 1801
chalup@gmx.net, maire@fit.qut.edu.au

**Abstract- This study empirically investigates variations of hill climbing algorithms for training artificial neural networks on the 5-bit parity classification task. The experiments compare the algorithms when they use different combinations of random number distributions, variations in the step size and changes of the neural networks' initial weight distribution. A hill climbing algorithm which uses inline search is proposed. In most experiments on the 5-bit parity task it performed better than simulated annealing and standard hill climbing.**

## 1 Introduction

Hill climbing is an optimisation algorithm which historically was first used for maximisation tasks. Like gradient descent algorithms in continuous spaces it approaches a local extremum but instead of using the gradient, hill climbing uses random local search to determine the direction and size of each new step. The terminology in the literature is not uniform. In this study hill climbing (HC) is the algorithm presented in Table 1 and hill climbing (type) algorithms are any of its variations shown in Tables 1 to 5.

Hill climbing type algorithms can be regarded as evolutionary algorithms for populations of size two which only use a mutation operator. They have been investigated extensively in the area of evolutionary computation, see e.g. [Ackley, 1987], [Chen and Flann, 1994] or [Yagiura and Ibaraki, 1996]. A comparison of genetic algorithms for neural network training and backpropagation is reported in [Kitano, 1990] and [Heistermann, 1994].

The most commonly used algorithms for training artificial neural networks are backpropagation type algorithms, see [Hertz et al., 1991]. Several versions have been developed. Some of them are fast (e.g. [Riedmiller and Braun, 1993]) and able to train large networks. Hill climbing algorithms are easier to implement and offer more flexibility in the training strategy and the neural network architecture. They have been used for (evolutionary) reinforcement learning tasks [Moriarty et al., 1999], [Pollack and Blair, 1998] and they are a worthwhile alternative in cases where gradient descent algorithms have been proved to be inadequate, e.g. training on long-term dependencies with recurrent neural networks [Bengio et al., 1994] or training networks which use non-differentiable activation functions.

This study shows how small modifications and adjustments to HC can cause qualitative changes to the convergence characteristics of a population of 40 neural networks when trained on the 5-bit parity task. Our aim is to provide better understanding and a heuristic for successful use of hill climbing type algorithms for training neural networks.

After describing the neural networks and the training task we explain several closely related hill climbing type algorithms. Then we analyse and discuss the training experiments.

## 2 Neural networks and training task

The training performance of a neural network depends on its architecture, the training data used, the distribution of the neural network's initial weights, the training algorithm and its parameters. For hill climbing essential parameters determine random variables (e.g. the components of the step matrix or the step size, see equations 1 and 2 in section 3). Once random variables are involved the type of random number generator or the computer precision can have influence on the training, too.

### 2.1 Neural networks

We used a fully-connected feed-forward architecture with two hidden units which had all possible shortcut and forward connections, see network (a) in Figure 1. This architecture has 18 connections, i.e. 6 more than the standard feed-forward network (which is network (d) in Figure 1). The underlying graph of all four networks (a)-(d) is directed and acyclic. All non-input units had a bias and a sigmoidal activation function. The architecture (a) has been found using an evolutionary optimisation algorithm which works on the topology of artificial neural networks, see [Yao and Liu, 1997]. Figure 2 shows training results which have been obtained with related neural network topologies.

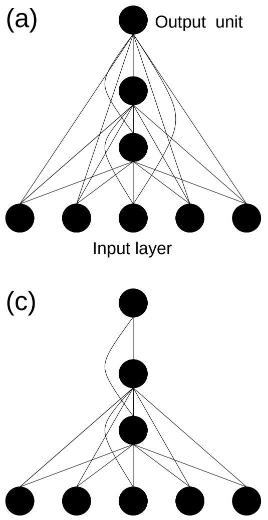


Figure 1: (a) Feed-forward network with two hidden units and all additional shortcut connections. (b) The connection between the two hidden nodes is pruned. (c) Standard feed-forward network with an additional link between the two hidden nodes. (d) Standard feed-forward network with two hidden units.

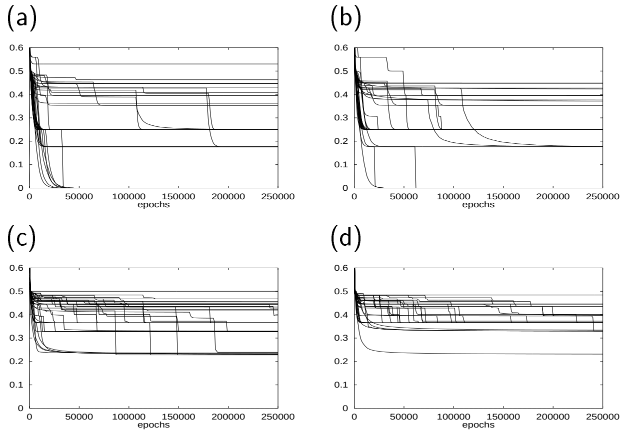


Figure 2: Training results corresponding to the four architectures in Figure 1. For each architecture 40 networks with different initial weights have been trained. The algorithm is (HC,2.0,C,C,0.001) with the notation convention of Figure 4. The ordinate is the root mean square error ($rmse$).

### 2.2 Training task

The training data consisted of the 32 patterns of the 5-bit parity task. In this task the network has to classify the binary input vectors into two classes: class 0 is formed by the vectors which have an even number of on-bits and class 1 contains the other vectors. The 5-bit parity task is a standard benchmarking task of medium difficulty for neural network training [Stork and Allen, 1992].

### 2.3 Random number distributions

HC uses random numbers at least at three steps (see pseudo code in Table 1 and explanations in section 3):

1. Generation of the initial weight matrix $W$.
2. Selection of a random number which determines the variable $stepsize$.
3. Calculation of the step matrix $dNW$ whose components are all new random numbers.

The simulated annealing algorithm requires a fourth random variable $U$ to decide if a new step is accepted or not, see Table 5.

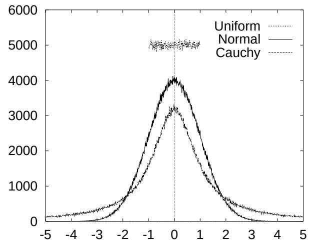


Figure 3: Histograms of 1,000,000 pseudo random numbers for each distribution (Uniform, Normal and Cauchy, from the top to the bottom) accumulated in 2000 bars.

We have used the algorithms 'ran1' and 'gasdev' from [Press et al., 1995] to be independent of a compiler or machine specific random number generator. They calculate pseudo random numbers which are uniformly or normally distributed, respectively. If $X, Y \in N(0,1)$ are two normally distributed random variables with 0 mean and standard deviation 1, a Cauchy distributed random variable [Rice, 1995] is obtained by taking the quotient $X/Y$ with $Y \neq 0$. The Cauchy distribution has heavier tails than the normal distribution, see Figure 3. Uniform random variables were taken from the interval (-1,1) and are denoted by $U \in U(-1,1)$. Normally distributed random variables have 0 mean and standard deviation 1 and are denoted by $N \in N(0,1)$. Derived Cauchy distributed random variables are denoted by $C \in C(0,1)$.

## 3 Training algorithms

The neural networks are trained in batch mode where the performance is measured by the root mean squared error ($rmse$) over the training set. The aim of the learning process is to minimise the $rmse$ within a maximum number of epochs (nEpochs).

All algorithms in this study have the basic structure of the *hill climbing* (HC) algorithm in Table 1. $W = (w_{ij})$ is the matrix of the neural network's weights. Each weight $w_{ij}$ connects unit $j$ to unit $i$. $NW$ is a new weight matrix ('the mutant') which is obtained by adding a randomly generated step matrix $dNW$ to $W$. Each coefficient of the step matrix is by default a Cauchy distributed random number $C$ whose standard deviation is scaled by the variable *stepsize*.

$$dNW := C * stepsize \tag{1}$$

The step size is a random variable $C$ which is adapted by the $rmse$ and scaled by two parameters $sf$ and $sl$.

$$stepsize := C * (sf * MIN(rmse, 1) + sl) \tag{2}$$

This method is motivated by [Angeline et al., 1994]. We chose the random variables $C$ by default Cauchy distributed but for comparison we have used normal and uniform probability distributions (section 4.2). The step size factor *sf* has been varied, see section 4.2, while the lower step size limit was left constant ($sl = 0.001$) in all experiments.

If the part of the pseudo code emphasised in bold italics in Table 1 is replaced by the code in Table 2 a variant of HC we call *inline hill climbing* (IHC) is obtained. IHC continues to use the same step matrix to proceed inline as long as it decreases the $rmse$ but at most for $il\_max$ steps. IHC with $il\_max = 1$ is equivalent to HC. During hill climbing training, often a smaller step size can increase the probability of finding a successful direction but at the same time it slows convergence. The IHC algorithm resolves this dilemma. Small steps are tested to find a good direction. Then efficiently the same small step is employed quickly several times in sequence effecting a large step. This can accelerate the training. There are several strategies for using the parameter $il\_max$ which restricts the number of inline steps. One strategy is to use $il\_max$ to limit inline exploitation on long ridges in favour of earlier exploration of better directions. On other tasks hill climbing strategies may be quite different and sometimes it might be wise to only allow very small mutations via dNW, compare [Pollack and Blair, 1998]. The *proportional inline hill climbing* (PIHC) algorithm in Table 3 is a refinement of IHC and takes some of these strategies into account. The parameter $il\_reach$ determines which fraction of each maximally possible change of $W$ is accepted.

If the code of Table 4 is substituted in Table 1 the algorithm becomes a *stochastic hill climbing* (SHC), if the code of Table 5 is substituted in Table 1 the algorithm becomes a *simulated annealing* (SA) algorithm [Kirkpatrick et al., 1983], [Dracopoulos, 1997], [Ingber, 1993]. Since SHC and HC essentially have the same structure we regard SHC and SA as hill climbing type algorithms which sometimes allow backward steps. SHC has a constant temperature while in SA the temperature is annealed. The temperature annealing schedule of SA is determined by an adaptive formula which depends on the $rmse$ or the epoch counter and on two parameters $sa1$ and $sa2$. SA or SHC with zero temperature (i.e. $sa1 = sa2 = 0$) are equivalent to HC. It can be shown [Aarts and Korst, 1989] that SA asymptotically always converges to a global minimum provided crucial parameters are set appropriately. For each task a suitable hill climbing algorithm can be adapted.

```
/* hill climbing algorithm (HC) */
  evaluate network with the initial weights W
  ⟶ rmse;
  WHILE((limit ≤ rmse) and (counter ≤ nEpochs))
     stepsize := C * (sf * MIN(rmse,1) + sl)
     FOR all weights of the neural net
        dNW := C * stepsize          /* step matrix */
        NW := W + dNW
     end FOR
     evaluate network with NW
     ⟶ rmse_new
     Δ := rmse_new - rmse
     IF(Δ < 0)
        W := NW          /* weight update */
        rmse := rmse_new
     end IF
     counter := counter + 1
  end WHILE
  return rmse_new
```

Table 1: Hill Climbing (HC).

```
WHILE(Δ < 0 and il_steps ≤ il_max)
   W := NW
   rmse := rmse_new
   NW := W + dNW
   evaluate network with NW
   ⟶ rmse_new
   il_steps := il_steps + 1
   counter := counter + 1
end WHILE
```

Table 2: Inline Hill Climbing (IHC). Substitute in Table 1.

## 4 Experiments

We report results from a comparative study of HC, IHC and SA and from experiments with HC in which different parameter settings are explored. The results are displayed in Table 6 and Figures 4 and 5. They reflect general tendencies and typical behaviour of the algorithms when trained on the 5-bit parity task. A different task might need a different parameter tuning.

| Algo- rithm | Distribution used for initial weights | step- size | step- matrix | sf | Nets with rmse $\leq$ 0.001 Epoch of earliest | Median of them | Numb. of | After 250,000 epochs Nets with 100% acc. | Average accuracy of the population |
|---|---|---|---|---|---|---|---|---|---|
| Experiments on combinations of random number distributions for step size and step matrix | | | | | | | | | |
| HC | N(0,1.0) | C | C | 0.001 | 29527 | 38455 | 3 | 3 | 92% (0.074 std) |
| HC | N(0,1.0) | C | N | 0.001 | 76862 | 103675 | 11 | 11 | 94% (0.072 std) |
| HC | N(0,1.0) | C | U | 0.001 | 121326 | 160106 | 8 | 8 | 93% (0.080 std) |
| HC | N(0,1.0) | N | C | 0.001 | 39779 | 45678 | 9 | 9 | 91% (0.116 std) |
| HC | N(0,1.0) | N | N | 0.001 | 194106 | 195016 | 2 | 8 | 93% (0.079 std) |
| HC | N(0,1.0) | N | U | 0.001 | - | - | 0 | 5 | 94% (0.061 std) |
| HC | N(0,1.0) | U | C | 0.001 | 50667 | 74240 | 8 | 8 | 91% (0.104 std) |
| HC | N(0,1.0) | U | N | 0.001 | - | - | 0 | 4 | 91% (0.091 std) |
| HC | N(0,1.0) | U | U | 0.001 | - | - | 0 | 1 | 92% (0.067 std) |
| HC | N(0,1.0) | U | U | 0.004 | - | - | 0 | 5 | 93% (0.062 std) |
| Experiments on comparison of HC, IHC and SA and increase of the step size factor *sf* | | | | | | | | | |
| HC | N(0,0.5) | C | C | 0.001 | 28040 | 40013 | 5 | 5 | 93% (0.072 std) |
| HC | N(0,0.5) | C | C | 0.01 | 27266 | 30998 | 6 | 6 | 94% (0.057 std) |
| HC | N(0,0.5) | C | C | 0.1 | 22910 | 34582 | 8 | 8 | 94% (0.067 std) |
| HC | N(0,0.5) | C | C | 0.5 | 33757 | 66975 | 5 | 5 | 91% (0.105 std) |
| IHC | N(0,0.5) | C | C | 0.001 | 25751 | 38043 | 14 | 14 | 95% (0.047 std) |
| IHC | N(0,0.5) | C | C | 0.01 | 24508 | 27432 | 9 | 9 | 94% (0.059 std) |
| IHC | N(0,0.5) | C | C | 0.1 | 20448 | 25450 | 8 | 8 | 92% (0.121 std) |
| IHC | N(0,0.5) | C | C | 0.5 | 27869 | 41502 | 4 | 4 | 88% (0.157 std) |
| SA | N(0,0.5) | C | C | 0.001 | 68276 | 159913 | 5 | 5 | 84% (0.147 std) |
| Experiments on variations in the range of the initial weights | | | | | | | | | |
| HC | N(0,0.1) | C | C | 0.001 | 41569 | 52915 | 3 | 4 | 85% (0.183 std) |
| HC | N(0,2.0) | C | C | 0.001 | 26091 | 35137 | 7 | 7 | 90% (0.117 std) |
| HC | N(0,3.0) | C | C | 0.001 | 21749 | 26564 | 3 | 3 | 89% (0.099 std) |
| HC | N(0,4.0) | C | C | 0.001 | 30778 | 34120 | 3 | 3 | 88% (0.104 std) |
| IHC | N(0,2.0) | C | C | 0.001 | 25229 | 31336 | 7 | 7 | 93% (0.060 std) |
| IHC | N(0,3.0) | C | C | 0.001 | 31522 | 84633 | 4 | 4 | 87% (0.112 std) |

Table 6: Overview of the experimental results (compare graphs in Figures 4 and 5 ).

```
dW = 0
WHILE(Δ < 0 and il_steps ≤ il_max)
    W := NW
    rmse := rmse_new
    NW := W + dNW
    evaluate network with NW
    ⟶ rmse_new
    il_steps := il_steps + 1
    dW := dW + dNW
end WHILE
if(il_steps > 2)
    W : = W - il_reach * dW
    evaluate network with W
    ⟶ rmse
```

Table 3: Proportional Inline Hill Climbing (PIHC). Substitute in Table 1.

```
IF(Δ < 0 or U < hc_temp)
    W := NW
    rmse := rmse_new
end IF
```

Table 4: Stochastic Hill Climbing (SHC). Substitute in Table 1.

```
sa_temp = MIN(rmse*sa1, (1-counter/nEpochs))*sa2))
IF(Δ < 0 or U < 1/(1+exp(Δ/sa_temp)))
    W := NW
    rmse := rmse_new
end IF
```

Table 5: Simulated Annealing (SA). Substitute in Table 1.

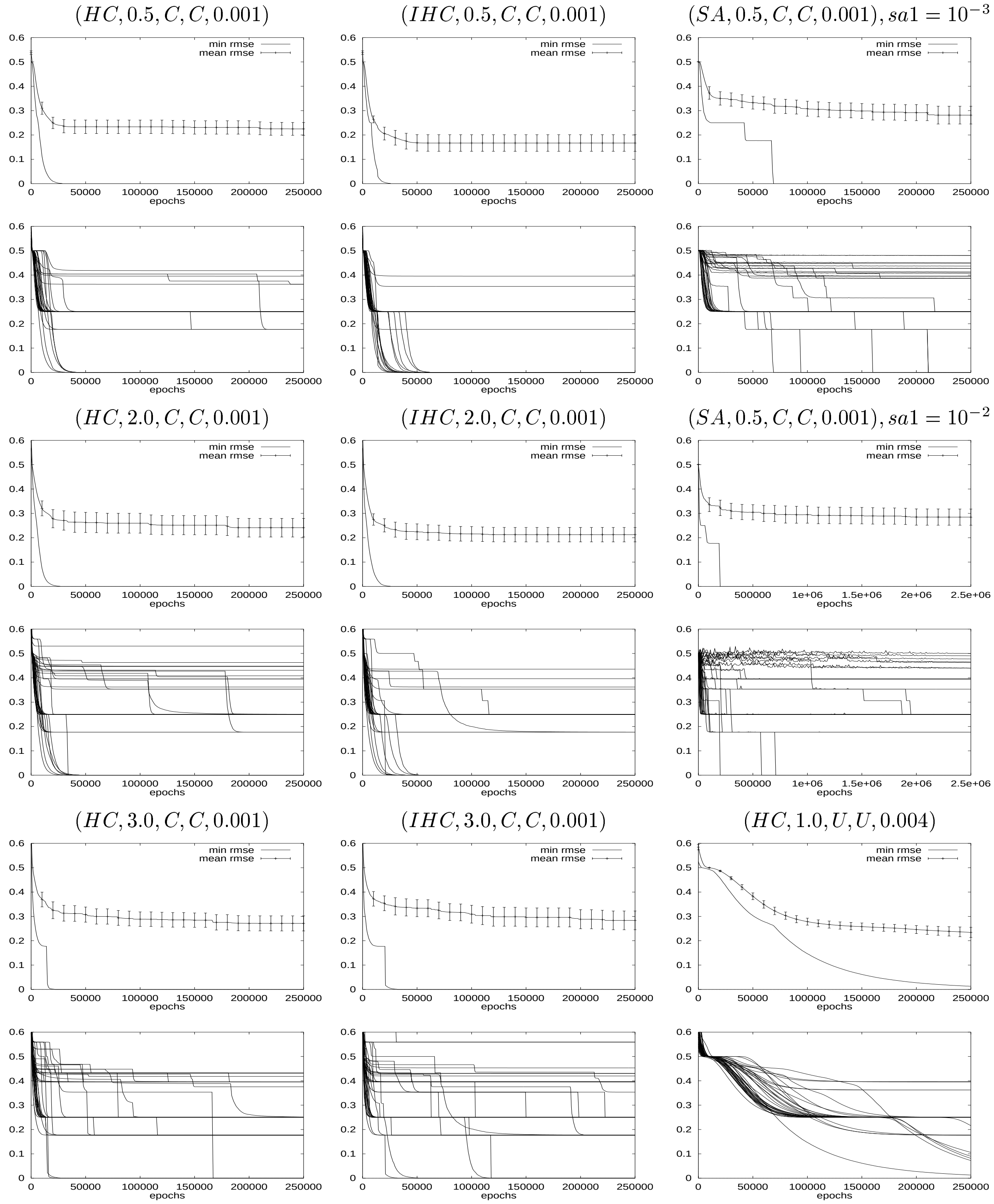


Figure 4: Results of experiments which compare HC, IHC and SA and evaluate variations of the range of the initial weights. Each of the nine experiments is displayed in two graphs. The upper graph shows the learning curve of the *min rmse* and the average learning curve of the population (*mean rmse*). The horizontal coordinate is the number of epochs and the ordinate is the *rmse*. The error bars indicate the confidence intervals for the mean of a normal distribution for a confidence level of 95%. The lower graph is the superposition of all 40 learning curves. Each experiment is labelled by a vector which specifies the following parameters *(type of algorithm, standard deviation of the initial weight distribution, type of distribution used for stepsize, type of distribution used for the step matrix, step size factor sf)*. Corresponding results are listed in Table 6.

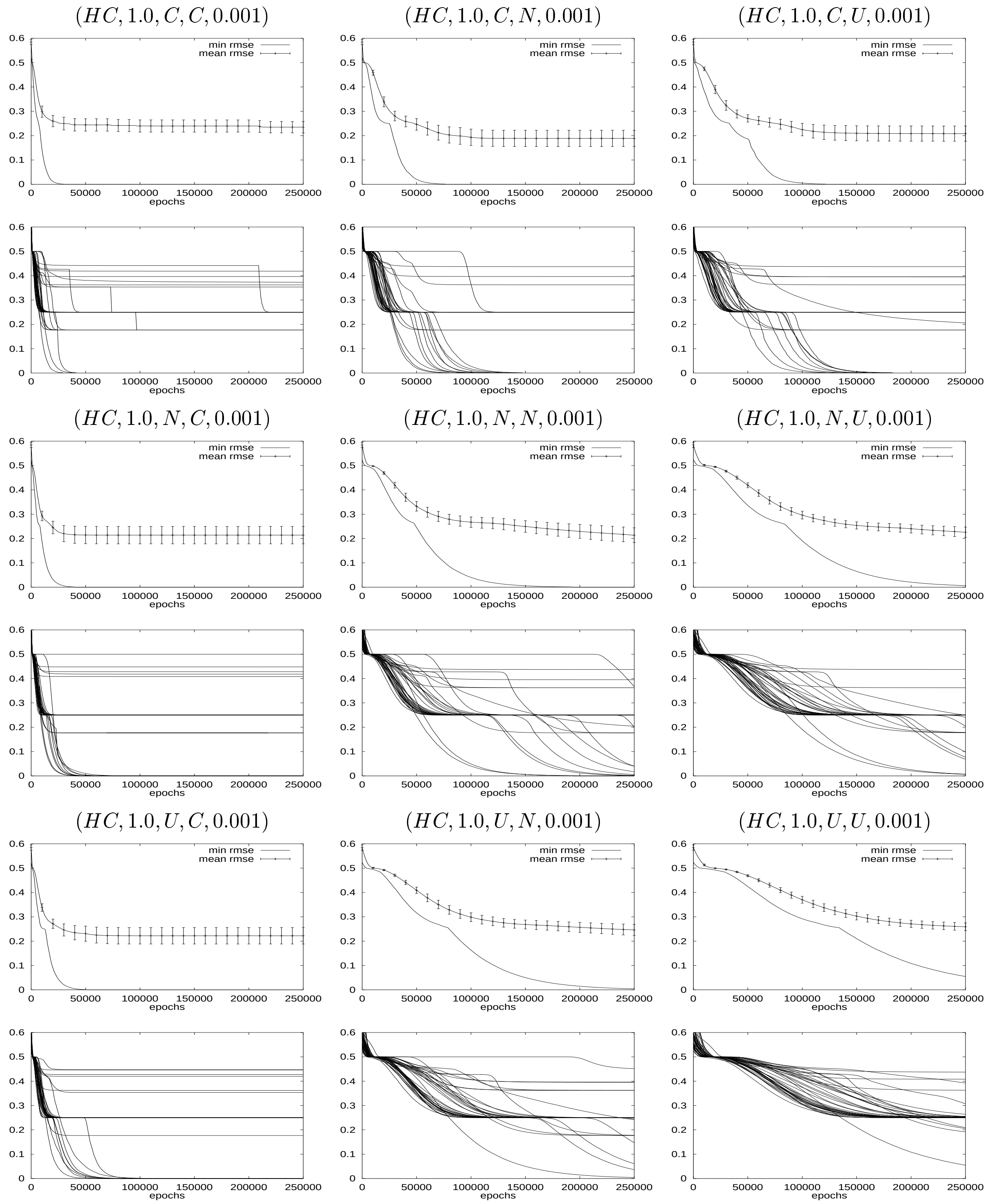


Figure 5: Results of nine experiments using different combinations of random number distributions for the generation of the step size and the coefficients of the step matrix. The display and the notation is explained in the commentary to Figure 4. The results are discussed in section 4.2 and listed in Table 6

After some preliminary study we settled on a number of fixed conditions for our simulations:

- The network architecture with two hidden units was kept constant, see Figure 1(a).
- After some initial tests with population sizes of 10, 20, ... ,100 networks we decided to use a constant population size of 40 networks. A comparison of several populations of size 40 and larger behaved essentially the same while results from populations with sizes smaller than 20 had a high variance.
- By default the weights were initialised with random numbers from normal distributions $N(0, 0.5)$ or $N(0, 1.0)$.
- Each network was trained until $rmse < 0.001$ with a maximum of 250,000 epochs.
- We calculated confidence intervals for the average of the 40 learning curves ($rmse$) of each experiment in Figures 4 and 5. The confidence level is set to 95%.

### 4.1 Comparison of HC, IHC and SA

In a comparative experiment IHC performed best of all three algorithms, see the first line in Figure 4. The IHC algorithm produced a network capable of successful classification most rapidly and the average learning curve was significantly below the learning curve of the standard HC. For these simulations always the value $il\_max = 10$ was employed. Tests with other parameter settings (see Table 6) supported IHC's advantage but the differences were not always significant.

Tests with SA were made using $\mathrm{sf} \in \{0.1, 0.01, 0.001\}$, $\mathrm{sa1} \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ and $\mathrm{sa2} = 0.5 \cdot \mathrm{sa1}$. With sufficiently low temperature SA performed closely to HC with the same parameters. If the temperature was increased SA could not improve on the performance of HC and the convergence of the networks was delayed or inhibited. Two example training results with $\mathrm{sa1} = 10^{-3}$ and $\mathrm{sa1} = 10^{-2}$ are displayed in Figure 4. In the second of the two experiments training was for 2,500,000 epochs.

In a flat landscape with narrow spikes, ramps or ridges SA's backward steps can kick the hill climber off a promising path which it can hardly reach again. SA has an advantage on other training tasks where small local minima are an issue or if the error landscape has a fractal characteristic [Lister, 1995].

### 4.2 Combinations of the random number distributions for step size and step matrix and the step size factor sf

Each of the formulas for generating the step size and the step matrix $dNW$ in HC uses a random variable C (see Table 1).

In a series of experiments we changed the distribution of these random variables from a Cauchy ($C$) to a normal ($N$) and a uniform distribution ($U$) (Figure 3).

Experimental results with all nine combinations of the three distributions are displayed in Figure 5. The fastest convergence was obtained using the Cauchy distribution for generating the random numbers needed for the step matrix, see column 1 in Figure 5.

The step size factor *sf* scales the standard deviation of the distributions. On average the networks converge faster when the stepsize factor *sf* is increased from 0.001 to 0.01, compare Table 6. We repeated all experiments of Figure 5 with $sf = 0.01$ instead of $sf = 0.001$. The convergence was faster but the relative differences of the experiments were maintained.

### 4.3 Initial weights

HC is sensitive to the choice of the initial weights when used for learning the 5-bit parity task. Hence we always trained a population of 40 networks, each of them starting from a different set of initial weights. In all experiments only a few networks of the population were able to learn the task. In a series of experiments we changed the range of the initial weights. Table 6, the first column in Figure 4 and the first graph in Figure 5 show results from training with weights initialised from a normal distribution with 0 mean and standard deviations of 0.1, 0.5, 1.0, 2.0, 3.0 and 4.0. Faster converging networks were found when the standard deviation of the initial weight distribution was increased to 3.0.

## 5 Conclusion

This study focused on tests with the 5-bit parity data, a medium difficult problem which offers good possibilities to observe algorithm behaviour and to investigate ways of tuning. The IHC algorithm performed best in our experiments though all algorithms were sensitive to the choice of the initial weights. SA seems not to be an appropriate algorithm for the 5-bit parity task. Good results with HC and IHC were obtained when the random components of the step size or the step matrix were Cauchy distributed. An increase of our initial settings for the step size and the range of the initial weights improved the performance.


## Acknowledgements

Thanks to Alan Blair, Lawrence Buckingham, Gerry Finn and Ross Hayward for helpful comments and to all colleagues at the FoD Machine Intelligence Workshop at Mullumbimbi and at the MLRC who participated in supporting discussions.

## Bibliography


[Aarts and Korst, 1989] Aarts, E. and Korst, J. (1989). *Simulated Annealing and Boltzmann Machines*. John Wiley, Chichester.

[Ackley, 1987] Ackley, D. H. (1987). *A Connectionist Machine for Genetic Hillclimbing*. Kluver Academic Publishers, Boston.

[Angeline et al., 1994] Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5:54–65.

[Arbib, 1995] Arbib, M. (1995). *The Handbook Of Brain Theory And Neural Networks*. MIT Press.

[Baryo, 1995] Baryo, A. G. (1995). Learning as hillclimbing in weight space. In [Arbib, 1995], pages 531–533.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

[Chen and Flann, 1994] Chen, I. and Flann, N. S. (1994). Parallel simulated annealing and genetic algorithms: a space of hybrid methods. Number 866 in Lecture Notes in Computer Science, pages 428–438. Springer-Verlag.

[Dracopoulos, 1997] Dracopoulos, D. C. (1997). *Evolutionary Learning Algorithms for Neural Adaptive Control*. Springer-Verlag.

[Heistermann, 1994] Heistermann, J. (1994). Different learning algorithms for neural networks - a comparative study. Number 866 in Lecture Notes in Computer Science, pages 386–396. Springer-Verlag.

[Hertz et al., 1991] Hertz, J., Krogh, A., and Palmer, G. (1991). *Introduction To The Theory Of Neural Computation*. Addison-Wesley.

[Ingber, 1993] Ingber, L. (1993). Simulated annealing: Practice versus theory. *Mathl. Comput. Modelling*, 18(11):29–57.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 222:671–680.

[Kitano, 1990] Kitano, H. (1990). Empirical studies on the speed of convergence of neural network training using genetic algorithms. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 789–795. MIT Press, Cambridge.

[Lister, 1995] Lister, R. (1995). Fractal strategies for neural network scaling. In [Arbib, 1995], pages 403–405.

[Moriarty et al., 1999] Moriarty, D., Schultz, A., and Grefenstette, J. (1999). Reinforcement learning through evolutionary computation. *Journal on Artificial Intelligence Research (JAIR)*. submitted.

[Pollack and Blair, 1998] Pollack, J. B. and Blair, A. (1998). Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32:225–240.

[Press et al., 1995] Press, W. H., Teukolsky, S. A., T., V. W., and Flannery, B. P. (1995). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition.

[Rice, 1995] Rice, J. A. (1995). *Mathematical Statistics and Data Analysis*. Duxbury Press, 2 edition.

[Riedmiller and Braun, 1993] Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In Ruspini, H., editor, *Proceedings of the IEEE International Conference on Neural Networks (ICNN), San Francisco*, pages 586–591.

[Stork and Allen, 1992] Stork, D. G. and Allen, J. D. (1992). How to solve the n-bit parity problem with two hidden units. *Neural Networks*, 5:923–926.

[Yagiura and Ibaraki, 1996] Yagiura, M. and Ibaraki, T. (1996). Metaheuristics as robust and simple optimization tools. In *Proceedings 1996 IEEE Conference on Evolutionary Computation*, pages 541–546.

[Yao and Liu, 1997] Yao, X. and Liu, Y. (1997). A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713.