Training Neural Nets with the Reactive Tabu Search

Roberto Battiti and Giampietro Tecchioli

***Abstract*— In this paper the task of training subsymbolic systems is considered as a combinatorial optimization problem and solved with the heuristic scheme of the reactive tabu search (RTS). An iterative optimization process based on a "modified local search" component is complemented with a meta-strategy to realize a discrete dynamical system that discourages limit cycles and the confinement of the search trajectory in a limited portion of the search space. The possible cycles are discouraged by prohibiting (i.e., making tabu) the execution of moves that reverse the ones applied in the most recent part of the search. The prohibition period is adapted in an automated way. The confinement is avoided and a proper exploration is obtained by activating a diversification strategy when too many configurations are repeated excessively often. The RTS method is applicable to nondifferentiable functions, is robust with respect to the random initialization, and effective in continuing the search after local minima. Three tests of the technique on feedforward and feedback systems are presented.**

I. Introduction

OPTIMIZATION is not sufficient for a successful learning scheme (consider, for example, the problems of generalization, noise-robustness, feature and example selection), but the minimization of a suitable "performance function" E often is a crucial component of learning. Derivative-based optimization methods like backpropagation (BP) [39] have been used with success in many practical contexts, but they stop at the first local minimum. When this happens, they can be restarted from a new random point, but in this case the previous work is wasted. In addition, the random initialization problem is not trivial [28] and the calculation of derivatives (provided that E is differentiable) is expensive and error-prone [40], especially if analog VLSI (very large scale integration) hardware is used.

In this paper a radically different approach to the learning task is presented. First the task is transformed into a combinatorial optimization problem so that the points of the search space are the vertices of a binary hypercube (or the set of binary strings with a specified length). Of course, in a digital computer each weight is represented by a fixed number of bits so that the problem is intrinsically combinatorial, although its nature is hidden from us by the floating point hardware and software. The problem is then solved with an heuristic method based on the construction of a search trajectory by a discrete dynamical system, with a dynamics designed to bias the motion toward points with low E values and to discourage the occurrence of limit cycles and the confinement in a limited portion of the search space. The bias is obtained by a modified local search component that evaluates a set of elementary moves applied to the current point (the neighborhood) and selects the best one. To this basic component one adds the prohibition of the inverses of recently executed moves, to discourage cycles, and a diversification strategy, to avoid the confinement of the solution trajectory.

In the presented application of the reactive tabu search (RTS) [6], the neighborhood of a point in the search space consists of the strings differing by a single bit, and the selected move is the one that causes the largest decrease in E among those that have not been already executed in the most recent part of the search. The prohibition period is regulated by a cycle-detection and reaction mechanism based on the previous history of the process. RTS escapes rapidly from local minimizers, it is applicable to nondifferentiable and even discontinuous functions, being based only on the availability of E values, and is very robust with respect to the choice of the initial configuration. In addition, the possibility of fine-tuning the number of bits for each parameter is useful to decrease the size of the search space, to increase the expected generalization, and to realize cost-effective VLSI devices.

The design criteria of the RTS technique for a search space given by binary strings are summarized in Section II, and the application of RTS to the training of subsymbolic systems is described in Section III. The results of two experimental tests on feedforward systems for classification tasks are discussed in Sections IV-A and IV-B. Finally, one test on a feedback system for nonlinear control is analyzed in Section IV-C.

II. Reactive Tabu Search: A Method Designed to Discourage Cycles

Let us define our notation. An instance of a combinatorial optimization (CO) problem [38] is a pair (F, E), where F is a set of feasible points with finite cardinality (we do not consider the case of a countably infinite set) and E is the cost function, i.e., a mapping: $E\colon F \to R^1$. A solution f is globally optimal if

$$E(f) \leq E(y) \quad \text{for all } y \in F.$$

For many interesting CO problems, the computational complexity for finding the globally optimal solution is not acceptable, so that one must resort to heuristic search methods for finding suboptimal points [18], [14]. The neighborhood function $N(f)$ associates to each point f a subset of F

$$N\colon F \to 2^F.$$

Manuscript received July 12, 1993; revised October 11, 1994. This work was upported in part by the INFN Iniziativa Specifica "RTS" and by the Progetto Speciale 1995, Dip. di Matematica, Univ. di Trento.

R. Battiti is with the Dipartimento di Matematica, Università di Trento and INFN, Gruppo Collegato di Trento, 38050 Povo (Trento), Italy.

G. Techiolli is with the Istituto per la Ricerca Scientifica e Tecnologica and INFN, Gruppo Collegato di Trento, 38050 Povo (Trento), Italy.

IEEE Log Number 9409368.

A point f is locally optimal with respect to N or a local minimizer if

$$E(f) \leq E(g) \quad \text{for all } g \in N(f).$$

The minimizer is strict if $E(f) < E(g)$. It is useful to define the neighborhood $N(f)$ as the set of points that can be obtained by applying to f a set of elementary moves $\mathcal{M}$

$$N(f) = \{g \in F \text{ such that } g = \mu(f) \text{ for } \mu \in \mathcal{M}\}.$$

In the present work, F is the set of all binary strings with a finite length L: $F = \{0,1\}^L$ and the elementary moves $\mu_i (i = 1, \cdots, L)$ change the ith bit of the string $f = [f_1, \cdots, f_i, \cdots, f_L]$

$$\mu_i([f_1, \cdots, f_i, \cdots, f_L]) = [f_1, \cdots, \bar{f}_i, \cdots, f_L] \tag{1}$$

where $\bar{f}_i$ is the negation of the ith bit: $\bar{f}_i \equiv (1 - f_i)$. Obviously, two moves commute and μ_i is idempotent (i.e., $\mu_i^2 = 1$, the identity move) and therefore its inverse is $\mu_i^{-1} = \mu_i$.

The TS strategy [20] has been used to solve a growing number of complex combinatorial optimization problems in an effective and efficient manner, mainly by the operations research community. To our best knowledge, the first applications of the standard TS for training an associative memory are in [2] and [42]. In [12] and [19] the tabu dynamics is adapted to produce a viable neural search technique, that remedies the one-shot descent offered by the Hopfield model [24]. In this work the framework is that of combinatorial optimization (CO) and the RTS discrete dynamics is not married with the continuous Hopfield network.

The TS scheme uses an iterative local search algorithm (like "steepest descent") to bias the search toward points with low E values. In addition, the TS incorporates strategies to avoid the occurrence of limit cycles.[1] The two goals are attained by using the following design principles

- *Modified Local Search.* At each step of the iterative process, the best move is selected from a set of admissible elementary moves that lead to points in the neighborhood of the current state. The best move is the one that produces the lowest value of the cost function E. Note that the best move is executed even if E increases with respect to the value at the current point, while the standard local search technique stops if the best move increases E. Increases are allowed because they are necessary to exit from local minimizers of E.
- *Cycle Avoidance.* The inverses of the moves executed in the most recent part of the search are prohibited (the names "tabu" or "taboo" derive from this prohibition).

In detail, at a given iteration t of the search, the set of moves $\mathcal{M}$ is partitioned into the set $\mathcal{T}^{(t)}$ of the tabu moves, the usefulness of which will become clear in what follows, and the set $\mathcal{A}^{(t)}$ of the admissible moves, i.e., of the moves that can be applied to the current point: $\mathcal{M}^{(t)} = \mathcal{A}^{(t)} \cup \mathcal{T}^{(t)}$; $\mathcal{A}^{(t)} \cap \mathcal{T}^{(t)} = \emptyset$.

At the beginning, the search starts from an initial configuration $f^{(0)}$, that can be generated randomly, and all moves are admissible: $\mathcal{A}^{(0)} = \mathcal{M}, \mathcal{T}^{(0)} = \emptyset$. The trajectory $f^{(t)}$ is then generated where the successor of the current point is obtained by a suitable move $\mu^{(t)}$ from the set $\mathcal{A}^{(t)}$. For example, if $\mathcal{A}^{(t)}$ contains a finite number of moves, one can select the best admissible move

$$f^{(t+1)} = \mu^{(t)}(f^{(t)}) \quad \text{where } \mu^{(t)} = \arg \min_{\nu \in \mathcal{A}^{(t)}} E(\nu(f^{(t)})).$$

If more moves cause the same E value, the move to apply is selected randomly from them. If the admissible moves are expensive to evaluate, for example if $\mathcal{A}^{(t)}$ is very large, one can sample $\mathcal{A}^{(t)}$ randomly and take the best out of $\mathcal{S}^{(t)} \subset \mathcal{A}^{(t)}$. The cardinality of the "sample" set $\mathcal{S}^{(t)}$ is called SAMPLE.

Let us now motivate the introduction of prohibited moves. In isolation, the above cited "modified local search" principle can generate limit cycles. Let us suppose that the current point $f^{(t)}$ is a strict local minimizer: the cost function at the next point must increase: $E(f^{(t+1)}) = E(\mu^{(t)}(f^{(t)})) > E(f^{(t)})$, and there is the possibility that the move at the next step will be its inverse $(\mu^{(t+1)} = \mu^{(t)^{-1}})$ so that the state after two steps will come back to the starting configuration

$$f^{(t+2)} = \mu^{(t+1)}(f^{(t+1)}) = \mu^{(t)^{-1}} \circ \mu^{(t)}(f^{(t)}) = f^{(t)}.$$

At this point, if the set of admissible moves is the same, the system will be "trapped" forever in a limit cycle of length 2. But this cycle is avoided if the inverses of the moves executed in the most recent part of the search are prohibited. The prohibition must be canceled after a certain number of iterations T because the tabu moves can be necessary to reach the optimum in a later phase of the search. The number of iterations T that a move remains in the $\mathcal{T}^{(t)}$ set is called "list size" in the original terminology, a term referring to a realization of the scheme in which the forbidden moves are inserted into a first-in first-out list (i.e., a queue of length T where a move enters as soon as it has been executed and exits after T steps). If the selection of a move is pictured as the firing of a neuron, the prohibition is a sort of refractory period for that neuron. In RTS the prohibition period $T^{(t)}$ is time-dependent and the set of prohibited moves is

$$\mathcal{T}^{(t)} = \{\mu \in \mathcal{M} \text{ such that its most recent use has been at time } \tau \geq (t - T^{(t)})\}. \tag{2}$$

A worked out example of the tabu search technique is presented in the Appendix. Let us mention that the following competing requirements hold:

- T must be large to avoid cycles. In detail, T must be larger than $(R/2) - 1$ to make cycles of length R impossible (note that R is even for binary strings).
- T must be sufficiently small to avoid over-constraining the trajectory, and in any case it must be smaller than or equal to $L - 2$.

A. The RTS Algorithm

The RTS algorithm [6] is here briefly summarized to understand its use in the area of subsymbolic machine learning studied in Section III. Because the longest possible cycle in the search space $F = \{0,1\}^L$ has length $R = 2^L$ (the list of the

[1] A search trajectory converges to a limit cycle if $f^{(t+nR)} = f^{(t)}, \forall t \geq t_0, n \in \mathcal{N}, n > 0$. R is the "repetition period," or "length" of the cycle.

Gray codes—see Section III—corresponding to the integers $0, 1, \cdots, 2^L - 1$ is an example of a cycle of maximum length obtainable with the elementary moves μ_i), the basic Tabu Search mechanism cannot guarantee the absence of cycles. In addition, the choice of a fixed T without *a priori* knowledge about the possible search trajectories that can be generated in a given (F, E) problem is difficult. If the search space possesses an inhomogeneous structure, a size T that is appropriate in a region of F may be inappropriate in other regions. For example, T can be too small to avoid cycles, or too large, so that only a small fraction of the movements are admissible and the search is inefficient.

RTS automatically changes the prohibition period $T^{(t)}$ during the search so that its value is appropriate to the local structure of the problem (basic reaction) and adopts a second-level reaction to deal with cycles that are not avoided by using the first reaction. The most recent iteration when each move μ_i has been applied is recorded and each configuration $f^{(t)}$ touched by the search trajectory is stored in memory with the most recent time when it was encountered. Let us introduce the functions:

- $\Lambda(\mu)$: the last iteration when μ has been used ($\Lambda(\mu) = -\infty$ if μ has never been used).
- $\Pi(f)$: the last iteration when f has been encountered ($\Pi(f) = -\infty$ if f has not been encountered).
- $\Phi(f)$: the number of repetitions of configuration f in the search trajectory ("repetition counter"). At the beginning $\Phi(f) = 0$ for all configurations.

There can be situations where f has been encountered but is not contained in the memory (in this case $\Pi(f) = -\infty$ and $\Phi(f) = 0$). In fact the allotted memory size can be insufficient or the algorithm can cancel the memory content at specific times (in particular see the function **diversify_search** of Fig. 3).

At iteration t, the set $\mathcal{A}^{(t)}$ contains the moves that have not been used in the most recent part of the trajectory

$$\mathcal{A}^{(t)} = \{\mu \in \mathcal{M} \text{ such that } \Lambda(\mu) < (t - T^{(t)})\}. \quad (3)$$

Note that checking the tabu status of a move requires only a couple of CPU cycles if the function $\Lambda(\mu)$ is realized with an array in memory.

Fig. 1 describes the main structure of the RTS algorithm.[2] The initialization part is followed by the main loop that continues to be executed until a satisfactory solution is found or a limiting number of iterations is reached. In this loop, the current configuration is compared with the previously visited points stored in the memory by calling the function **memory_based_reaction** (Fig. 2) that returns two possible values (DO_NOT_ESCAPE or ESCAPE). In the first case the next move is selected by calling **best_move** (Fig. 3); in the other case the algorithm enters a diversification phase based on a short random walk, see the function **diversify_search**

```
(Initialize the data structures for tabu:)
$t \leftarrow 0$                          (iteration counter)
$T^{(0)} \leftarrow 1$                    (prohibition period)
$t_T \leftarrow 0$                        (last time $T$ was changed)
$\mathcal{C} \leftarrow \emptyset$        (set of often-repeated configurations)
$R_{ave} \leftarrow 1$                    (moving average of repetition interval)
$f^{(0)} \leftarrow$ random $f \in F$     (initial configuration)
$f_b \leftarrow f^{(0)}$                  (best so far $f$)
$E_b \leftarrow E(f^{(0)})$               (best so far $E$)


repeat

    (See whether the current configuration is a repetition:)
    escape ← memory_based_reaction($f^{(t)}$)   (see Fig.2)
    if escape = DO_NOT_ESCAPE then
        $\mu \leftarrow$ best_move   (see Fig. 3)
        $f^{(t+1)} = \mu(f^{(t)})$
        $\Lambda(\mu) \leftarrow t$
        (Update time, and best_so_far:)
        $t \leftarrow (t+1)$
        if $E(f^{(t)}) < E_b$ then
            $E_b \leftarrow E(f^{(t)})$
            $f_b \leftarrow f^{(t)}$
    else
        diversify_search   (see Fig.3)
until $E_b$ is acceptable or maximum iteration reached
```

Fig. 1. RTS: main structure.

(Fig. 3). For each new configuration on the trajectory, the lowest E value found during the search is saved with the associated configuration f, because otherwise this point could be lost when the trajectory escapes from a local minimizer. The couple (f_b, E_b) is the suboptimal solution provided by the algorithm when it terminates.

In machine learning applications, it is useful to terminate the search when the generalization is maximal to avoid over-training the system. In the applications presented in Sections IV-A, IV-B, and IV-C, the expected generalization will be estimated on a validation set (called test set).

B. Reactive Schemes of RTS

The reactive mechanisms of the algorithm modify the discrete dynamical system that defines the trajectory so that limit cycles and confinements (that can be compared to chaotic attractors in dynamical systems) are discouraged. The reaction is based on the past history of the search, and it causes possible changes of $T^{(t)}$ or the activation of a diversifying phase. Short limit cycles are avoided by modifying $T^{(t)}$. In particular, see the function **memory_based_reaction** defined in Fig. 2, the current configuration f is compared with the configurations visited previously and stored in memory. If f is found, its last visit time $\Pi(f)$ and repetition counter $\Phi(f)$ are updated. Then, if its repetition count is greater than the threshold REP, f is included into the set $\mathcal{C}$, and if the size $|\mathcal{C}|$ is greater than the threshold CHAOS, the function returns immediately with the

[2]The structure of the program is illustrated with simple selection (conditional) and iteration keywords (**if** $\cdots$ **then** $\cdots$ **else**, **repeat**), the assignment operator ($X \leftarrow Y$ means that the value of variable X is overwritten with the value of Y) and functions that can return values to the calling routine. Compound statements are indented, function names are in boldface, and comments are in italics.

```
function memory_based_reaction($f$)
comment: The function returns ESCAPE when an escape action is to be executed, DO_NOT_ESCAPE
otherwise. INCREASE = 1.1, DECREASE = 0.9, CHAOS = REP = 3.
    Search for configuration $f$ in the memory:
    if $\Pi(f) > -\infty$ then ($f$ visited previously)
        Find the cycle length, update last_time and repetitions:
        $R \leftarrow t - \Pi(f)$   ($R$ = time interval)
        $\Pi(f) \leftarrow t$
        $\Phi(f) \leftarrow \Phi(f) + 1$   ($\Phi(f)$ = repetitions of $f$)
        if $\Phi(f) >$ REP then
            $\mathcal{C} \leftarrow \mathcal{C} \cup f$   (add $f$ to set of often-repeated config.)
            if $|\mathcal{C}| >$ CHAOS then
                $\mathcal{C} \leftarrow \emptyset$
                return ESCAPE (reaction III)
        if $R < 2(L-1)$ then (if cycle is avoidable)
            $R_{ave} \leftarrow 0.1 \times R + 0.9 \times R_{ave}$
            $T^{(t+1)} \leftarrow Min(T^{(t)} \times \text{INCREASE}, L-2)$ (reaction I)
            $t_T \leftarrow t$
    else
        (If the configuration is not found, install it:)
        $\Pi(f) \leftarrow t$
        $\Phi(f) \leftarrow 1$

    if $(t - t_T) > R_{ave}$ then
        $T^{(t+1)} \leftarrow Max(T^{(t)} \times \text{DECREASE}, 1)$ (reaction II)
        $t_T \leftarrow t$
    return DO_NOT_ESCAPE
```

Fig. 2. RTS: The function **memory_based_reaction**.

```
function best_move
comment: The function returns the move to be applied to the current configuration.
If SAMPLE $< |\mathcal{A}^{(t)}|$ a subset of the admissible moves is tested.
    $\mathcal{S} \leftarrow \{Max(\text{SAMPLE}, |\mathcal{A}^{(t)}|)$ moves randomly extracted out of $\mathcal{A}^{(t)}\}$
                                (if SAMPLE $\geq |\mathcal{A}^{(t)}|$, all moves are taken)
    $\mu \leftarrow \arg\min_{\nu \in \mathcal{S}} E(\nu(f^{(t)}))$

    return $\mu$
function diversify_search
comment: The function executes a sequence of random steps, that become tabu as soon as they are
applied.
    Clean the memory structure $\Pi$ and $\phi$
    $\mathcal{S} \leftarrow \{Min(1 + R_{ave}/2, |\mathcal{M}|)$ moves randomly sampled out of $\mathcal{M}\}$
    repeat for $\sigma \in \mathcal{S}$
        $f^{(t+1)} \leftarrow \sigma(f^{(t)})$
        $\Lambda(\sigma) \leftarrow t$
        (Update time, and best_so_far:)
        $t \leftarrow (t+1)$
        if $E(f^{(t)}) < E_b$ then
            $E_b \leftarrow E(f^{(t)})$
            $f_b \leftarrow f^{(t)}$
```

Fig. 3. RTS: The functions **best_move** and **diversify_search**.

value ESCAPE. If the repetition interval R is sufficiently short (if $R < 2(L-1)$), one can discourage cycles by increasing $T^{(t)}$ in the following way: $T^{(t+1)} \leftarrow T^{(t)} \times$ INCREASE. Precisely, the largest T that leaves at least two admissible moves is $T =$

$L-2$, so that only cycles of length $R < 2(T+1) = 2(L-1)$ can be safely avoided by using the tabu set $\mathcal{T}$. At least two moves must always be admissible (otherwise the move is not influenced by the E values), so that an upper bound of $L-2$ is set on $T^{(t)}$ (this explains the "Min" operator in reaction I).

If f is not found, it is stored in memory, the most recent time when it was encountered is recorded ($\Pi(f) \leftarrow t$) and its repetition counter is set to one ($\Phi(f) \leftarrow 1$).

If T is not allowed to decrease, its value value will remain large after a phase of the search with many repetitions, even in later phases, when a smaller value would be sufficient to avoid short cycles. Therefore, $T^{(t)}$ is reduced by the factor DECREASE < 1 if it remains constant for a number of iterations greater than the moving average of repetition intervals R_{ave} (reaction II in Fig. 2).

The best move can be selected either by testing all admissible moves or by sampling a subset of them. The two possibilities are selected with the parameter SAMPLE: all moves in $\mathcal{A}^{(t)}$ are tested if SAMPLE $\geq |\mathcal{A}^{(t)}|$ (in particular if SAMPLE $= \infty$), otherwise only SAMPLE different moves are randomly extracted from $\mathcal{A}^{(t)}$ and tested.

When the first-level reactions that modifies $T^{(t)}$ (reactions I and II in Fig. 2) are not sufficient to guarantee that the trajectory is not confined in a limited portion of the search space, the search dynamics enter a phase of "random walk" (reaction III in Fig. 2 and function **diversify_search** of Fig. 3). The number of random steps is proportional to the moving average R_{ave}, the rationale being that more steps are necessary to escape from a region that causes long cycles. Note that the execution time of the random steps is registered ($\Lambda(\sigma) \leftarrow t$), so that they become tabu; see (3). When the "random walk" phase begins, the memory structure is cleaned but this is not equivalent to a random restart because R_{ave} and $T^{(t)}$ are not changed, and, when this phase terminates, the prohibition of the most recent random steps discourages the trajectory from returning into the old region.

In passing, let us note that the space and time complexity of the reaction scheme amounts to some bytes and to a small and approximately constant number of machine cycles per iteration, provided that a compressed version of the configuration is stored and that the hashing mechanism is used for obtaining the values $\Pi(f)$ and $\phi(f)$. In the hashing scheme f is stored in a memory location whose address is a function address = *hash* (f). The number of possible addresses N_a must be larger than the maximum number of items to store N_i (say $N_a > 2N_i$) and the *hash*() function must "scatter" the addresses of different f's so that the probability that two of them obtain the same address is small. In the straightforward application presented, the compressed information is simply the floating point value of E for the given configuration. The choice is effective if the probability that two different configurations have the same E value is small.

Let us note that the tabu search dynamics is designed to explore the search space in an efficient way. It can be demonstrated that the probability of visiting points at large Hamming distances with respect to a starting configuration is much higher than in the case of a random walk in the search space. The RTS algorithm is studied in detail in [6], while [5] is dedicated to a study of the parallel properties. Benchmarks and comparisons with respect to simulated annealing [9], repeated local minima search, genetic algorithms, and "mean field theory" neural nets [10] have been executed with fully satisfactory results.

III. The Application for Training Neural Nets

We consider two paradigmatic systems in the area of neural networks: the multi-layer perceptron (MLP) (see the applications considered in Sections IV-A–IV-B), and the recurrent neural network of [43] (see Section IV-C). The notation for the MLP system and the tranformation into a combinatorial optimization task are described in this section; the feedback system considered will be illustrated in Section IV-C.

Input units of an MLP are denoted by I_i, "hidden" units by H_i (we consider a single hidden layer) and output units by O_i. The parameters of the system are denoted by w_{jk} (weights between input and hidden layer) and by W_{ij} (weights between hidden and output layer). The function that maps an input pattern $\mathbf{I}$, whose components are real numbers, into the associated output vector $\mathbf{O}$ is constructed as follows. First the "net input" h_j of the hidden units is computed

$$h_j = \sum_k w_{jk} I_k. \tag{4}$$

Threshold values are incorporated by fixing the activation value of one unit in the input and one unit in the hidden layer to one. Then the activation H_j is obtained by using a "sigmoidal" function

$$H_j = \frac{1}{1+e^{-h_j}}. \tag{5}$$

At the next layer, first the net input for the output units is computed as

$$o_i = \sum_j W_{ij} H_j. \tag{6}$$

and, finally, the output is obtained as follows

$$O_i = \frac{1}{1+e^{-o_i}} \tag{7}$$

The system is trained by using a set of P example patterns (i.e., of associations between input $\mathbf{I}_p$ and desired output $\mathbf{D}(\mathbf{I}_p)$) and by minimizing with respect to $\mathbf{w}$ the usual sum-of-squared-errors measure

$$E(\mathbf{w}) = \sum_{p,i} (O_i(\mathbf{w}, \mathbf{I}_p) - D_i(\mathbf{I}_p))^2. \tag{8}$$

Each weight of the network is described by a binary string of B_w bits. The B_w bits are the Gray code of an integer in $[0, 2^{B_w} - 1]$. The Gray code has the property that the nearby integers $n-1$ and $n+1$ are obtained by changing a single bit of the code of n (i.e., the codes of $n+1$ and $n-1$ have a Hamming distance of one with respect to the code of n). The conversions between the binary encoding $b_{B_w} b_{B_w - 1} \cdots b_1$ and

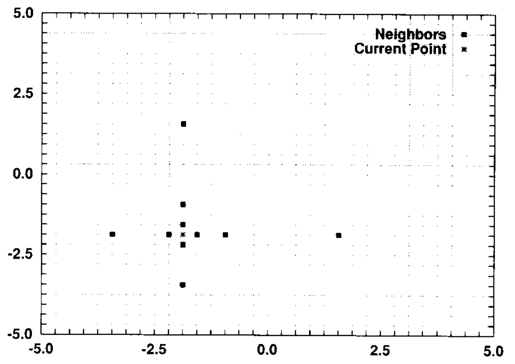


Fig. 4. Neighborhood in the X-Y plane obtained with the Gray encoding.

the Gray encoding $g_{B_w} g_{B_w-1} \cdots g_1$ are as follows (see, for example, [16])

$$\begin{cases} g_k = b_k & \text{if } k = B_w \\ g_k = b_{k+1} \oplus b_k & \text{if } k < B_w \end{cases} \tag{9}$$

$$\begin{cases} b_k = g_k & \text{if } k = B_w \\ b_k = b_{k+1} \oplus g_k & \text{if } k < B_w \end{cases} \tag{10}$$

where $\oplus$ is the exclusive-or operator and the second transformation must be done for decreasing values of k, starting from $k = B_w$. When the network is evaluated, the Gray code for each weight is transformed into the base-two binary code of a positive integer n and, finally, into a floating point value w in the range $[-W_s/2, +W_s/2]$, as follows

$$w = \left(\frac{n}{2^{B_w} - 1} - \frac{1}{2} \right) W_s. \tag{11}$$

If the memory is sufficient, the conversion can be executed with a lookup table. The binary string for the optimization algorithm is obtained by concatenating the Gray codes for the weights.

Given a weight w, by changing one of the B_w bits in the encoding (and by repeating the operation for all possible bits), one obtains B_w weights in the neighborhood. For the cited property of the Gray code, the neighborhood contains the nearest weights on the discretized grid, plus a cloud of points at growing distances in weight space.

In Fig. 4 we show a typical neighborhood in the $X - Y$ plane obtained with $B_w = 5$, so that the binary string for the (x, y) point has 10 bits, and with $W_s = 10$, so that each coordinate ranges from -5.0 to 5.0. The reachable points are at the intersections of the grid lines; the 10 neighbors of the point $(x = -1.875, y = -1.875)$ are illustrated with boxes. Note that the property that nearest points in w-space are always contained in the neighborhood does not hold if the standard base-two binary encoding is used. For example, the integer coded with the 10-bit binary string [1000000000] follows the integer coded with [0111111111], but more than one basic move is necessary to accomplish the transition in the binary string (in fact 10 moves are needed in this case). Because the neighborhood contains the nearest points, RTS can reproduce a discretized form of steepest descent (BP) if the additional neighbors do not provide better E values. The proper choice of the encoding and of the associated elementary moves are critical to the success of the method.

A. Neighborhood Evaluation

The evaluation of the points in the neighborhood requires much less CPU time than the evaluation of the same number of arbitrary points. For the MLP neural net, an efficient scheme is based on storing all the intermediate results of the forward pass for the current configuration. When a single weight is modified because of a change in one of its bits, the change is propagated forward in a very fast manner.

Let us consider networks with a single hidden layer. The intermediate values for the network corresponding to the current configuration are saved when the output corresponding to a given input pattern is calculated (let us call these values $h\_current_j$ and $o\_current_i$).

When a basic move is executed, only one bit of the string is changed, and therefore a single weight of the network is modified. Let us distinguish the case of a change in a first-layer weight (Δw_{jk}) from that of a change in a second-layer weight (ΔW_{ij}). In the first case, if Δw_{jk} is the change in a weight in the first layer, (4) is used to find the change in the "net input" of the affected neuron in the hidden layer

$$h_j \leftarrow h\_current_j + \Delta w_{jk} I_k \tag{12}$$

and then the activation H_j is calculated using (5). Because partial derivatives are not needed, the squashing function does not need to be differentiable and a realization with a lookup-table is sufficient even if the table size is limited by the available memory space (a table with 2^{16} entries is sufficient for many classification tasks). If the "sigmoidal" function is calculated with a lookup-table, the calculation of the activation from the net input requires only a couple of machine cycles. Finally the activations o_i of the output units are obtained as

$$o_i \leftarrow o\_current_i + W_{ij} \Delta H_j \tag{13}$$

where $\Delta H_j = H_j - H\_current_j$, and the outputs O_i are obtained with a second table-lookup.

If the change is in the second layer (ΔW_{ij}), the computation is even faster: o_i is updated as

$$o_i \leftarrow o\_current_i + \Delta W_{ij} H_j \tag{14}$$

and the outputs O_i are calculated with a single table lookup.

Let us summarize the average cost for evaluating the neighborhood for a network with N_I input, N_H hidden, and N_O output units, assuming that all weights have the same number of bits and therefore each weight has the same probability of being changed. Because there are $N_H \times N_I$ weights connecting the input to the hidden layer and $N_O \times N_H$ weights connecting the hidden to the output layer, the operations of (12) and (13) are needed SAMPLE $\times$ $N_I/(N_O + N_I)$ times, while the operations of (14) are needed SAMPLE $\times$ $N_O/(N_O + N_I)$ times, on the average. Therefore, if A and M are the CPU times required by a single addition (or subtraction) and multiplication and C_σ is the time required by the transfer function (possibly realized by the lookup-table), the average total computational complexity C_N for each example pattern is

$$\begin{aligned} C_N &= \text{SAMPLE}\frac{N_I}{(N_I+N_O)} \\ &\quad \times (2A+M+C_\sigma+N_O(A+M+C_\sigma)) \\ &\quad + \text{SAMPLE}\frac{N_O}{(N_I+N_O)}(A+M+C_\sigma). \qquad (15) \end{aligned}$$

Let us consider the case of a single output unit (as in the case of a two-class discrimination) and a large number of input units. After keeping only the dominating terms in the sum one obtains

$$C_N \approx \text{SAMPLE}(3A+2M+2C_\sigma). \qquad (16)$$

This result can be compared with the asymptotic approximation

$$C_N \approx N_H(N_I(A+M)+C_\sigma) \qquad (17)$$

for the feedforward pass of the MLP net that can be larger than (16) in the case of a large number of input and hidden units (in particular if $N_H \times N_I \gg$ SAMPLE, a common case in the applications illustrated in Section IV).

Because some MLP nets that are relevant for the applications have a large number of weights (say more than 1000) it is crucial to reduce the number of bits per weight as much as possible and to employ the partial evaluation of the neighborhood that was explained in Section II (by setting SAMPLE $\ll L$). In addition to reducing the time complexity of evaluating the neighborhood, a compact encoding of the network helps in reaching higher generalization performances.

IV. Experimental Tests

Two tests of RTS are presented for the MLP feedforward neural net and one for a feedback net used in a nonlinear control application. The first test (Section IV-A) is the XOR problem. The MLP neural net has a single hidden layer with two units and the same "squashing" function as the one used in [39]; see (5) and (7). Although the significance of the problem for automated learning is dubious (in fact the parity problem is such that patterns differing by a single bit produce opposite outputs), the task is paradigmatic for the presence of local minima (see, for example, [13]), and for the sensitive dependence of learning on the initial conditions [28].

The second problem considered (Section IV-B) is a benchmark task derived from a real-world application in experimental high energy physics (HEP), where a classifier with an MLP structure is used to discriminate patterns derived from a collision in the large electron-positron (LEP) collider into two classes: "background noise" or "potentially relevant event" related to the "bottom quark." The classifier is trained with examples of "background noise" patterns derived from the experimental setup and examples of "bottom" patterns derived from a simulator.

The last task (Section IV-C) is a version of the "truck and trailer backup" problem [34]. The controller is a discrete-time fully recurrent network that defines the backup motion of the truck when it is initialized randomly in a given region. Contrary to the two previous tasks, in this case the only information used for the training is the final positions and orientations reached by the truck-and-trailer system and the possible violation of constraints (there is no teacher giving the correct control variable for a given configuration).

To eliminate the possible effects of a coarse lookup table, in all tests we use the standard double-precision "squashing function" $\sigma(h_i) = 1/(1+e^{-h_i})$ (shifted with the subtraction of 0.5 for the control task). The initial binary string at iteration zero of RTS is generated by randomly setting each bit with equal probability for the values one and zero.

A. The XOR Function

To estimate the effects of the number of bits per weight and the neighborhood sample, a total of 100 tests per data point were executed, by varying the B_w parameter (two, four, and eight bits) and the SAMPLE value. The size of the complete neighborhood is $B_w \times 9, 9$ being the total number of weights, including thresholds.

The network has two hidden units (2-2-1), the W_s parameter is equal to 20, and a training session is terminated when all patterns are less than 0.2 away from the target. In Fig. 5 we show the average number of RTS iterations (with its standard deviation) as a function of the number of points in the neighborhood (SAMPLE). The three curves are for growing numbers of bits per weight. It can be observed that the average number of steps decreases rapidly in passing from SAMPLE $= 2$ to SAMPLE $= 5$ (for $B_w = 2, 4$), and slowly afterwards, while it reaches a minimum at around SAMPLE $= 4$ for $B_w = 8$. The CPU time is approximately proportional to the number of configurations evaluated, (see Fig. 5 (bottom)), and it reaches the minimum for a small SAMPLE value, approximately SAMPLE $= 7$ for $B_w = 2, 4$ and SAMPLE $= 4$ for $B_w = 8$. For this problem two bits are sufficient, but four bits produce a faster convergence, while eight bits are excessive and slow down the search; see the curve marked with diamonds in Fig. 5(b).

As it is expected, RTS is very effective in escaping from local minima and in continuing the search until the desired solution is reached. In all tests the algorithm converges in 100% of the cases, although a large number of local minima are encountered during the search. For example, the XOR problem was run with two bits per weight and with the complete neighborhood evaluation, and the number of local minima encountered was counted (i.e., the number of configurations such that all elementary moves produced a strictly higher E value). On the average (100 training tests), seven local minima were encountered during each search. The average number of TABU steps is 86.8, so that the frequency of encounter is approximately of one local minimum every 12 steps.

A comparison with on-line BP, with learning rate $= 0.1$, momentum $= 0.0$, was executed by varying the initial scale W_i chosen for randomizing the weights and by counting the number of successes (for a maximum of 10^6 iterations). The average number of iterations for convergence (for the successful cases) and its standard deviation are listed in the last column of Table I. Our results are qualitatively similar to

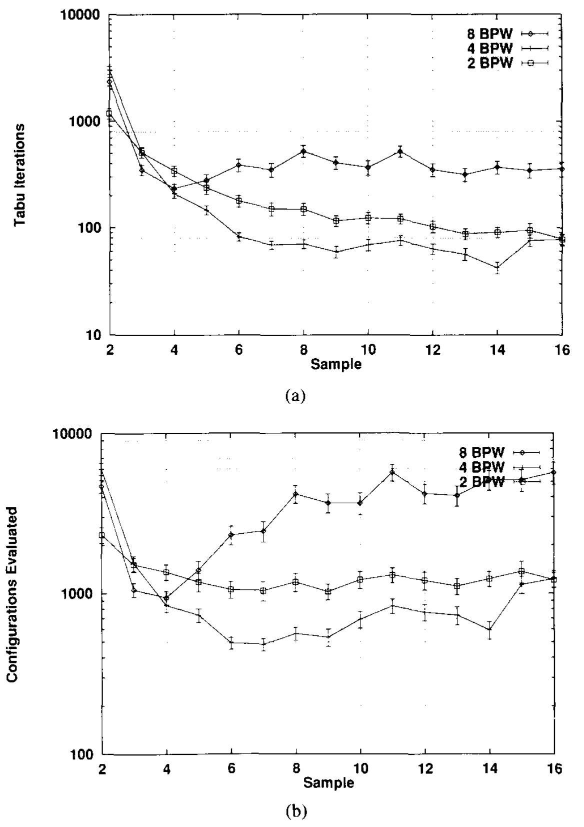


Fig. 5. (a) Average number of RTS steps and standard deviation of the average for the XOR problem, as a function of the number of points in the neighborhood. (b) Average number of points evaluated and standard deviation of the average. The three curves correspond to experiments with a growing number of bits per weight: $B_w = 2, B_w = 4, B_w = 8$.

TABLE I
ON-LINE BACKPROPAGATION FOR THE XOR PROBLEM (MAX. 10^6 ITERATIONS): SUCCESSES AND AVERAGE NUMBER OF ITERATIONS

initial scale W_i	successes in 100	on-line iterations (std.dev.)
0.1	75	45014.7 (14631.7)
0.5	89	10360.7 (562.1)
1.0	91	12530.8 (3802.4)
10.0	52	37426.9 (18268.2)
20.0	31	91603.2 (31993.3)

those of [28]: a careful selection of the starting configuration is critical to the success of BP. In particular, when the initial scale is the one corresponding to the initialization of the TABU algorithm ($W_s = 20$), BP converges only in 31 cases out of 100, with a very large average number of iterations.

It should be remembered that on-line BP has been used effectively by developers on a series of significant applications, so that the above problems related to the initialization or to the presence of local minima should not discourage its usage (see also [23] for a discussion of cases where local minima are absent). A proper initialization is also critical to the success of "batch" BP. Table 2 lists the performance results of batch BP with constant learning rate $= 0.1$.

TABLE II
BATCH BP FOR THE XOR PROBLEM (MAX. 10^6 ITERATIONS): SUCCESSES AND AVERAGE NUMBER OF ITERATIONS

initial scale W_i	successes in 100	batch iterations (std.dev.)
0.1	67	484259.2 (215590.0)
0.5	98	44856.1 (49929.2)
1.0	96	16239.5 (12842.3)
-10.0	64	23370.0 (77689.8)
20.0	32	89302.7 (158133.0)

Available techniques to increase the "safety requirements" and the speed of convergence of BP, are, for example, the use of adaptive learning rates for on-line BP [30], the use of "line searches," and second-order information for batch BP; see [7] and [3] and the references contained.

B. Event Discrimination in High Energy Physics

Experimental HEP facilities need state-of-the-art discrimination systems for selecting and classifying the relevant events. In a typical facility, colliding particles produce streams of secondary particles—called "jets"—that leave traces in a large number of spatially arranged detectors. The frequency of events (that include "false alarms" and spurious signals) is so high that the registration of the event parameters onto secondary storage has to be restricted only to a subset of "potentially relevant" events. To this end on-line "triggering" mechanisms estimate the interest of the event, so that only the events whose estimated interest is greater than a selected threshold are registered.

MLP's are being used as pattern classifiers for the triggering, possibly with analog VLSI or dedicated hardware implementations. In particular neural nets have been proposed for discriminating bottom quark jets at LEP, the large electron-positron collider at the CERN laboratories [35].

In this paper we consider the task of recognizing two-jet events produced by the bottom quark as a benchmark for the RTS algorithm, both because of its applicative interest and because the large number of events available permits a statistically significant test of the relative performance of different algorithms. The same benchmark task has been used in [7] for comparing different training algorithms: i) the BP algorithm [39], ii) a version of gradient descent with adaptive step, iii) the conjugate-gradient technique, iv) the one-step secant method with fast line searches [3], and v) two versions of the stochastic search technique of [41], [15] (in particular the new proposal called affine shaker).

The patterns used for training and testing the neural classifiers have been produced with the COJETS event generator [36], using the natural frequencies. A total of 100 000 e^+e^- events have been generated at center-of-mass-system energy of 91 Gev. Of these only two-jet events have been retained, and among these only the jets with at least four particles have been selected. Each jet is described by a pattern with 17 features [35].

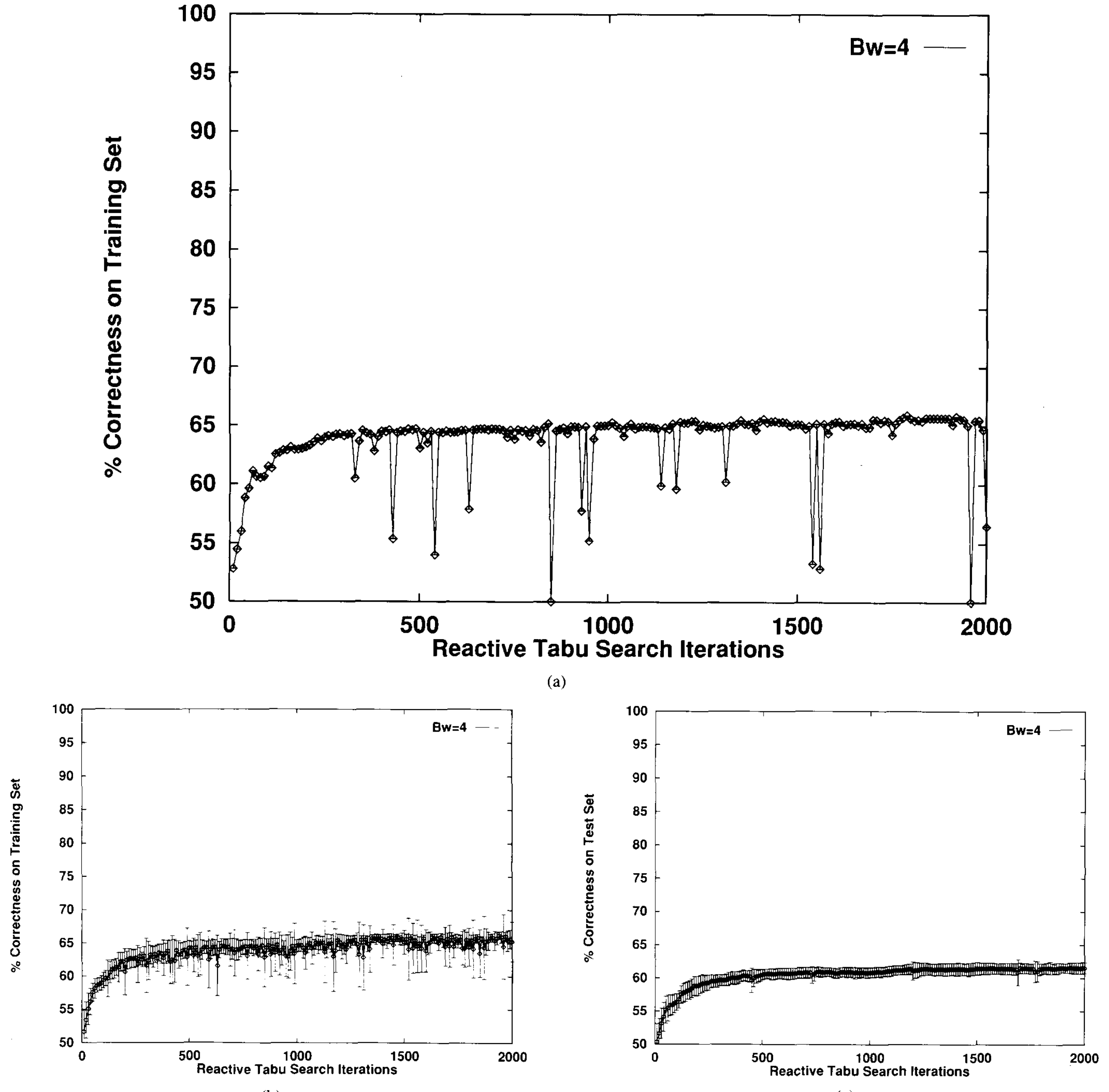


Fig. 6. Event discrimination in HEP ($B_w = 4$). (a) Typical case with correctness on training set, (b) Average training correctness, and (c) average generalization results.

The jets originated by the bottom quark have been subdivided randomly into two equal sets with 7 741 events each, to be used for training and testing, respectively. Similarly, the jets produced by the other quarks ("background") have been subdivided into two sets of 25 463 events each. Finally, the training set for the benchmark task has been obtained by selecting randomly 5000 "bottom" patterns and 5000 "background" patterns from two of the above sets, while a testing set with the same number of patterns has been obtained randomly from the other two sets. The desired output value is one for "bottom," zero for "background." When the generalization is measured, patterns with output value greater than 0.5 are classified as "bottom," while those with output less than or equal to 0.5 are classified as "background."

The MLP network architecture has 17 input, 10 hidden, and one output unit. The total number of weights (including the thresholds) is 191, and the length of the binary string for the presented tests ranges from $L = 191$ (for $B_w = 1$) to $L = 764$ (for $B_w = 4$). The scale parameter is $W_s = 10$, the number of points in the neighborhood that are sampled is SAMPLE $= 8$.

Three series of tests with $B_w = 1, 2$, and 4 have been executed. For each number of bits per weight, 10 different

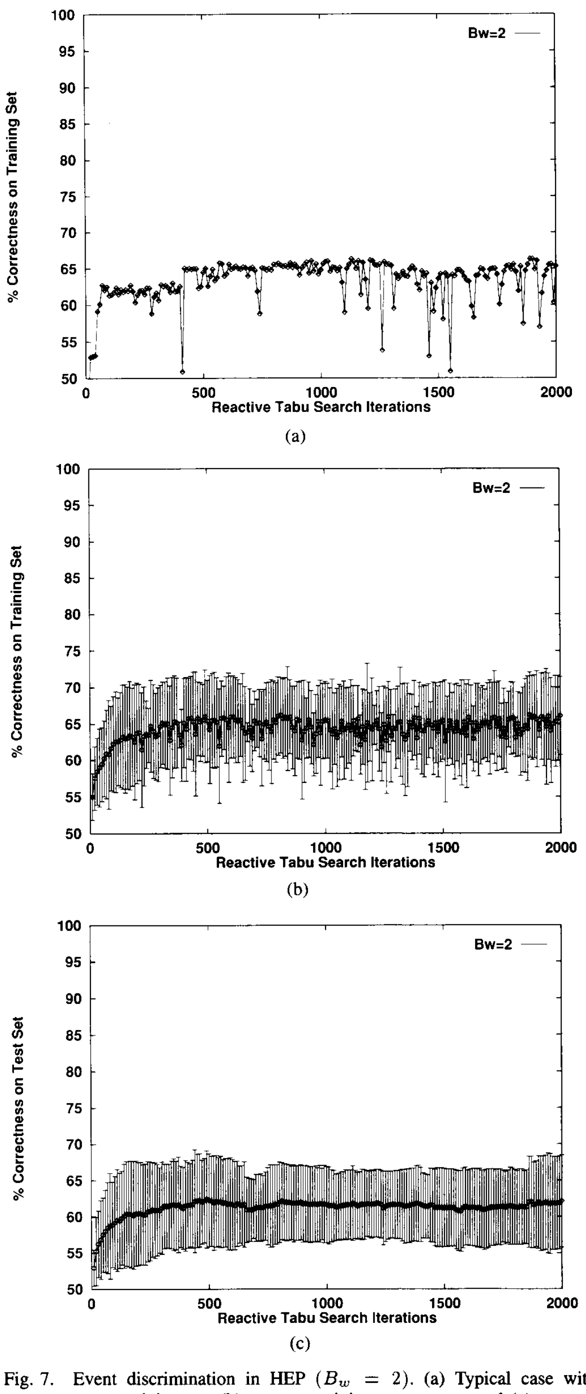


Fig. 7. Event discrimination in HEP ($B_w = 2$). (a) Typical case with correctness on training set, (b) average training correctness, and (c) average generalization results.

training sessions are completed after starting from different random initial points. To avoid "over-training," RTS is stopped when the maximum generalization levels are reached (2000 RTS iterations are sufficient).

For each series we present three plots. The first plot illustrates the evolution of the training session for one typical case (out of the 10 tests), by showing the percent of correct recognition obtained when the current network is tested on the training set. The correctness is checked at each step. The second plot shows the correctness on the training set (checked every 10 RTS iterations) averaged over the 10 tests, with the standard deviation σ of the correctness distribution at each checkpoint. The third plot shows the average generalization performance (i.e., the percent of correct recognition obtained by testing the current network on the testing set).

Fig. 6 shows the results for the case with $B_w = 4$. The performance on the training set starts at the 50% level (corresponding to a random classification), reaches the 62% level during the first hundreds of iterations, and then it tends to grow in a slower manner. Note that the performance curve on the training set does not increase monotonically, as it is clear from the jumps in the top curve of Fig. 6 in some cases the best admissible move is such that the performance decreases (see also Section II-A). The average generalization result reaches a "plateau" at about 1000 RTS iterations. The best generalization results are in good agreement with those obtained in [7] with continuous weight values and a selection of optimization techniques. This result indicates that the approximation capabilities of nets with four bits per weight are in this case sufficient to reach performances that are obtainable with "floating point" weights represented with 64 bits. Let us recall that, in a practical application, the threshold for the acceptance of a potential "bottom" event can be higher than 0.5, so that more "bottom" events are classified as "background" and rejected, but the surviving events have a larger probability of being true "bottom" events.

Finally, Fig. 8 presents the results obtained with $B_w = 1$, so that the weights have two possible values: -5.0 or 5.0. the standard deviation of the correctness results is larger, and the brisk moves in weight-space are evident by the performance jumps in the typical case (top curve of Fig. 8). The plateaus are produced by the exploration of a suboptimal region, while sudden jumps indicate that a new suboptimal region is explored (let us recall that RTS is designed to escape from local minimizers).

The average generalization results obtained with $B_w = 1$ are excellent if compared with the best results obtained in [7] for the case of weights with continuous values trained with continuous optimization techniques. The highest correctness level obtained with all the methods used in the cited paper is about 62% ($\text{std.dev.} = 0.5\%$). Although nets with continuous weights (represented by 64-bits floating point values) contain as a special case nets with the two above possible values, during the extensive series of experiments with continuous values such highly-accurate nets were not produced. The superior performance of the RTS algorithm can be explained by the following facts:

- *Different initialization*: The weights of nets trained with the reactive tabu search are initialized with a uniform distribution on the binary strings, corresponding to a uniform distribution on the discretized weights in the range $[-W_s/2, W_s/2]$, while the nets with continuous-valued weights in [7] are initialized with small random values (with a uniform distribution in the range $[-0.5, 0.5]$).

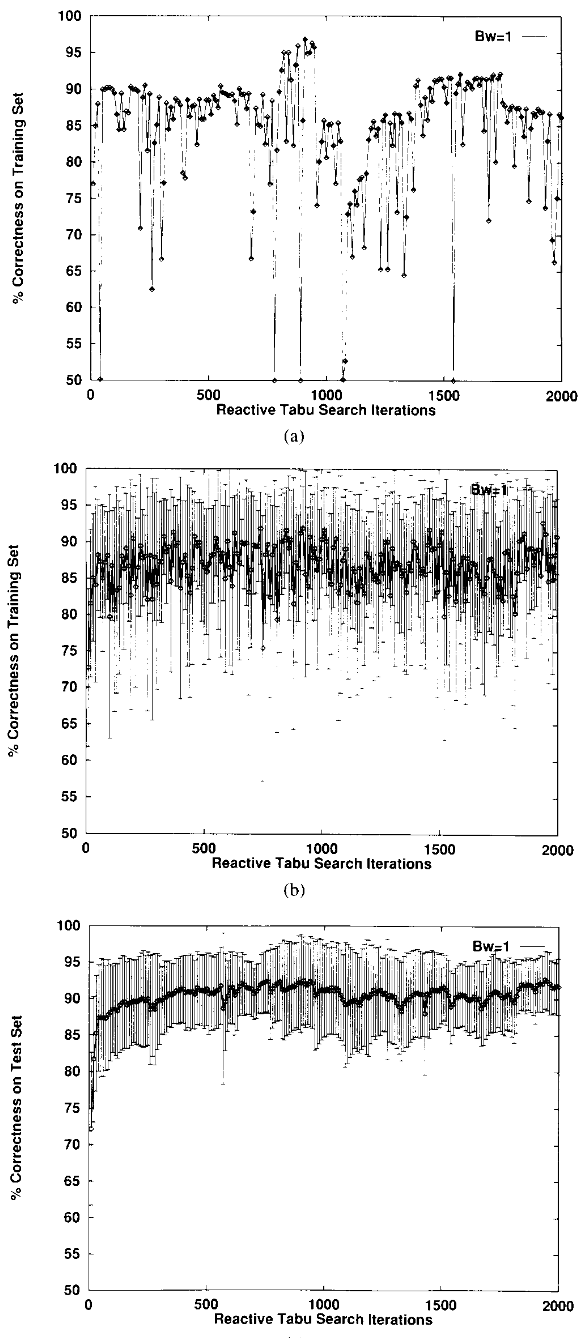


Fig. 8. Event discrimination in HEP ($B_w = 1$). (a) Typical case with correctness on training set, (b) average training correctness, and (c) average generalization results.

In fact, if the initial random weights are too large, the partial derivatives are exponentially small and the Hessian matrix is seriously ill-conditioned [40]. Experimentally, after starting from values in $[-W_s/2, W_s/2]$ for the derivative-based training techniques we could not obtain better results than those described in [7].

- *Different dynamics in the search space*: Techniques like BP base the step on a local model of the E function (first terms in the Taylor series expansion by using first and, in some cases, second partial derivatives), while the RTS technique base the step on a sample of the binary neighborhood that, because of the Gray coding, is translated into a cloud of points at different Euclidean distances in weight space; see Fig. 4. In addition, the RTS dynamics is different from a pure steepest-descent dynamics (see for example the capability of escaping from local minima). These facts imply that the trajectories in the weight-space produced by the different techniques are qualitatively different.
- *Wider exploration of the search space*: The ratio between the number of points that can be visited by RTS in the allotted CPU time and the total number of points in the search space decreases like $2^{-B_w N_w}$: a small number of bits permits a more effective exploration.
- *Reduced over-training*: The limited search space for $B_w = 1$ reduces the over-training effect, so that better generalization results can be expected if the network remains capable of "storing" the example associations.

C. Parking a Truck Behind Another Truck

In the task of steering a tractor-trailer truck backing up at constant speed [1], [34], the front wheels of the cab move a fixed distance backward at each step. The control signal is the angle u of the front tires with respect to the axis of the cab, and the goal considered in [34] is to guide the back of the trailer to a point on a loading dock with the trailer perpendicular to the dock.

In a backup trial the cab starts in a random position and orientation with respect to the dock, with a random angle between the cab and the trailer. Each trial terminates either when a part of the truck touches the edges of the parking space (the exact space occupation of the cab and trailer is checked at each step for a possible constraint violation) or when a maximum number of steps are executed (max. 256 in our tests).

The task is representative of many "sequential decision" problems: control decisions made early in the backing up process have substantial effects upon the final results. A truck backup task is considered in [29] for a comparison of neural and "fuzzy" systems. In [29] the MLP net is trained by using the examples (state, control variable) generated by the "fuzzy" system, a different task from the problem that we consider in which the error signal is generated only at the end of the entire backup sequence (there is no "expert driver" to guide the learning network).

The state variables of the truck illustrated in Fig. 9 are:

- x, y: coordinates of center of rear of trailer,
- θ_s: angle of trailer measured from positive x with counterclockwise being positive (radians), and
- θ_c: angle of cab, measured from positive x with counterclockwise being positive (radians).

The constraint on the possible configurations is that the cab cannot be rotated by more than $\pi/2$ degrees with respect to

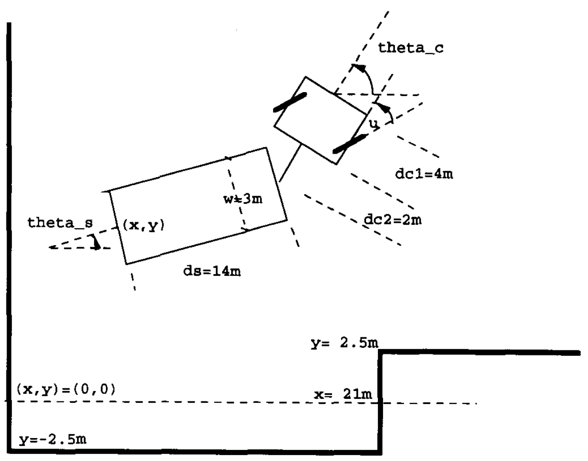


Fig. 9. State variables and desired final configuration for a tractor-trailer truck backing up at constant speed.

the trailer

$$|\theta_s - \theta_c| \le \pi/2. \tag{18}$$

The performance function is the sum of two terms $E = E_f + E_c$. The first term E_f is related to the desired final configuration $(x_{\text{dock}}, y_{\text{dock}}, \theta_{s\text{dock}} = 0)$

$$E_f = \alpha(x_{\text{dock}} - x[t_f])^2 + \beta(y_{\text{dock}} - y[t_f])^2 + (\theta_s[t_f])^2 \tag{19}$$

where t_f is the time steps at which the trial is terminated and the constants α and β regulate the relative importanceof the positional errors with respect to the error in the trailer angle ($\theta_s = 0$ means that the trailer is perpendicular to the loading dock). The second term E_c is proportional to the amount of violation of constraint (18)

$$E_c = \sum_{t=0}^{t_f} \gamma\Theta(|\theta_s[t] - \theta_c[t]| - \pi/2)(|\theta_s[t] - \theta_c[t]| - \pi/2) \tag{20}$$

where $\Theta(s)$ is the Heaviside function ($\Theta(s) = 1$ if $s > 0, 0$ otherwise), and the parameter γ regulates the compromise between constraint satisfaction and correct final configuration of the truck.

The control variable is u, the steering angle of front wheels with respect to cab orientation, counterclockwise positive (radians). The allowed range is $|u| \le (7/18)\pi$.

The truck dimensions are $d_c = 6$ m (cab length from pivot to front axle) $d_s = 14$ m (trailer length), and the fixed distance one that the front tires move in one time step is $r = 1$ m.

The kinematics of the truck is described by the following equations

$$A = r\cos u[t] \tag{21}$$
$$B = A\cos(\theta_c[t] - \theta_s[t]) \tag{22}$$
$$C = A\sin(\theta_c[t] - \theta_s[t]) \tag{23}$$
$$x[t+1] = x[t] - B\cos\theta_s[t] \tag{24}$$
$$y[t+1] = y[t] - B\sin\theta_s[t] \tag{25}$$

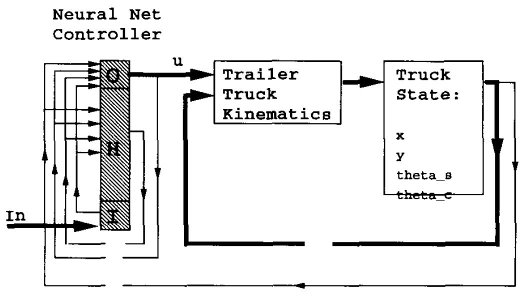


Fig. 10. Truck and trailer system. The free parameters are the weights of the feedback loops illustrated with thin lines. Δ delays the signal by one time step.

$$\theta_c[t+1] = \arctan\left(\frac{d_c\sin\theta_c[t] - r\cos\theta_c[t]\sin u[t]}{d_c\cos\theta_c[t] + r\sin\theta_c[t]\sin u[t]}\right) \tag{26}$$

$$\theta_s[t+1] = \arctan\left(\frac{d_s\sin\theta_s[t] - C\cos\theta_s[t]}{d_s\cos\theta_s[t] + C\sin\theta_s[t]}\right) \tag{27}$$

where arctan is from $-\pi$ to π.

The complete truck system illustrated in Fig. 10 is a recurrent network whose state is described by an array of variables c_i. It is useful to partition the indexes of these state variables into four sets: the set O of indexes of "output" variables (proportional to the single control signal u in the given application), the set I of "external input" variables, the set H of "hidden" variables, and the set T of the state variables of the truck $(x, y, \theta_s, \theta_c)$. At time step t, the control subsystem is updated as follows

$$\forall s \in \{H \cup O\}\ \text{sum}_s(t) \leftarrow \sum_{t \in I \cup H \cup O \cup T} w_{st}c_t(t) \tag{28}$$

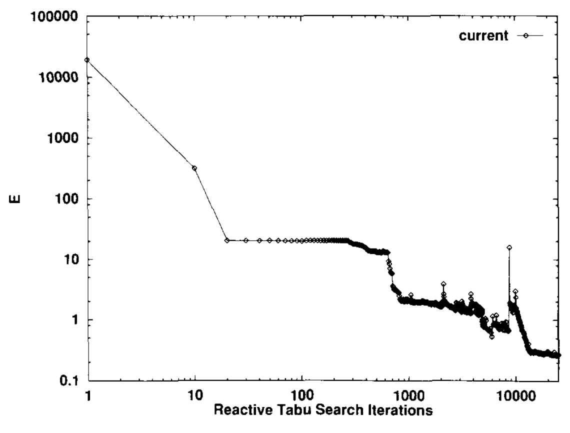
$$\forall s \in \{H \cup O\}\ c_s(t+1) \leftarrow \sigma_s(\text{sum}_s(t)) = -\frac{1}{2} + \frac{1}{1 + e^{-\text{sum}_s(t)}} \tag{29}$$

where $\text{sum}_s(t)$ denotes the net imput to the sth unit. Although the "squashing function" σ could be different for different units, in the present task we found that a single function with a range that is symmetric with respect to zero is sufficient to realize the control network. The control signal u (in radians) is obtained by multiplying the output variable c_o to scale it in the range corresponding to $[-70, 70]$ degrees

$$u(t+1) \leftarrow c_o(t+1) \times \pi \times \frac{7}{9}.$$

Finally, the state of the truck $(x, y, \theta_s, \theta_c)$ is updated by considering the current state and the value of the control variable u, through the kinematics equations described in (21)–(27). In our case the single external input is "clumped" to 1, as a convenient way to obtain a threshold for the activation of each state variable of the control subsystem.

The results obtained for the parking task defined in [34] (the truck with trailer is moving freely in the half-plane with positive x values) have been described in [8]. In this paper we present the results for the harder case described in Fig. 9. We place additional constraints in the moving space corresponding to the real-world task of parking a truck behind other trucks. In this case the simple solution of [26] is not applicable.



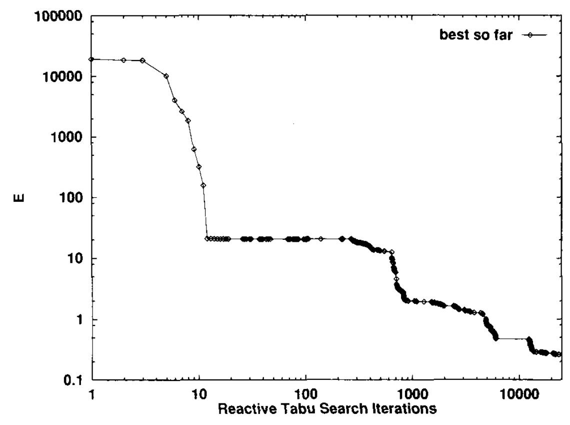


Fig. 11. Evolution of E during training (truck and trailer parking in constrained space). (a) Current E at a series of checkpoints. (b) "Best so far" values. Log-log scale.

In the top plot of Fig. 11 we show the values of the current E at selected checkpoints, while in the bottom plot we show the evolution of the "best so far" performance (E_b in Fig. 1). In the second graph a point is plotted as soon as a system configuration with lower E value is found. The large E values at the beginning are caused by the violation of the constraint $|\theta_s - \theta_c| \leq \pi/2$. The later evolution shows "plateau" regions, followed by performance jumps. As usual, the evolution of E is nonmonotonic because the exploration properties of RTS provoke a transition to different "attraction basins" of the search space (but clearly the "best so far" point is saved so that it can be used at the end of the training period).

The truck trajectories obtained at selected checkpoints are illustrated in Fig. 12, where we show the position of the point (x, y) (the "license plate" point) during the backup test. At $t = 0$, all 64 parking trials are unsatisfactory: either the constraints are violated or the cab oscillates in a wild manner and moves a short distance during the maximum number of backup steps. At $t = 10\,000$ most backup trials end up at the left wall, although the final position and orientation is far from the requirements. Some backup tests are stopped because the trailer touches the border of the moving space; see the trajectories that terminate near the dashed line in Fig. 12, the position of the entire truck is not shown for readability. For a growing iteration number all truck trajectories reach the left wall and the final position and orientation become progressively nearer to the values desired; see the situation at $t = 15\,000$ and $t = 25\,000$.

The trajectories obtained at RTS $= 25\,000$ correspond to realistic "expert driver" trajectories, like the one illustrated in Fig. 13: in the first backup steps the trailer is pushed away from the upper wall of the constrained space and prepared for the final phase. In the final steps the trailer is pushed inside the desired parking space, and the cab is turned rapidly to permit a close fit.

The parameters are $B_w = 8, W_s = 10$. The control system has six hidden units ($N_H = 6$), so that the total number of weights is 84 and the length of the binary string is $L = 672$. Sixty-four starting configurations for the truck-and-trailer system are generated randomly with a uniform distribution in the following range: $x[0] \in [40 \text{ m}, 60 \text{ m}], y[0] \in [6 \text{ m}, 8 \text{ m}], \theta_c[0] \in [-6°, 6°], \theta_s[0] \in [-6°, 6°]$.

For this task a constant value SAMPLE $= 20$ is sufficient but some CPU time can be spared by using a smaller sample at the beginning of the search and larger sample in the following phases. In particular, for the following results, the size of the sample neighborhood is SAMPLE $= 0.01 \times L = 6$ in the first phase of the search, SAMPLE $= 0.02 \times L = 13$ after 5000 iterations, and SAMPLE $= 0.03 \times L = 20$ after 20 000 iterations.

In the illustrated test, the function E corresponds to a large penalty for constraint-violation, and to a more strict requirement on the y coordinate with respect to the x coordinate ($\alpha = 1, \beta = 16, \gamma = 100$). In Fig. 11 we show the evolution of the performance function defined in (19) and (20). E is normalized by the number of backups used for training (64 backup for each evaluation).

V. Related Approaches

Related approaches, although within different frameworks, are the use of simulated annealing (SA) [27] for training MLP nets, (see, for example, [17]), and the use of genetic algorithms (GA's) [22] for optimizing weights and neural architectures, (see, for example, [44]). Space limitations do not permit a detailed discussion and comparison between RTS and the different versions of GA and SA. As a very brief remark, let us recall that simulated annealing is based on a connection between statistical mechanics and CO: random moves are generated from the current point, a move is always accepted if E decreases, while it is accepted with a probability $p \approx \exp(-\Delta E/\tau)$ if E increases. The "escape" from local minimizers is obtained in a stochastic manner, but, if the "temperature" τ is much lower than the height of a barrier around a local minimum, SA will spend an enormous time in its neighborhood before escaping. On the contrary, RTS is deterministic (if the complete neighborhood is evaluated) and the choice of the move depends on the past history of the search. The desired properties of the search trajectory are obtained by complementing the "greedy" component with the

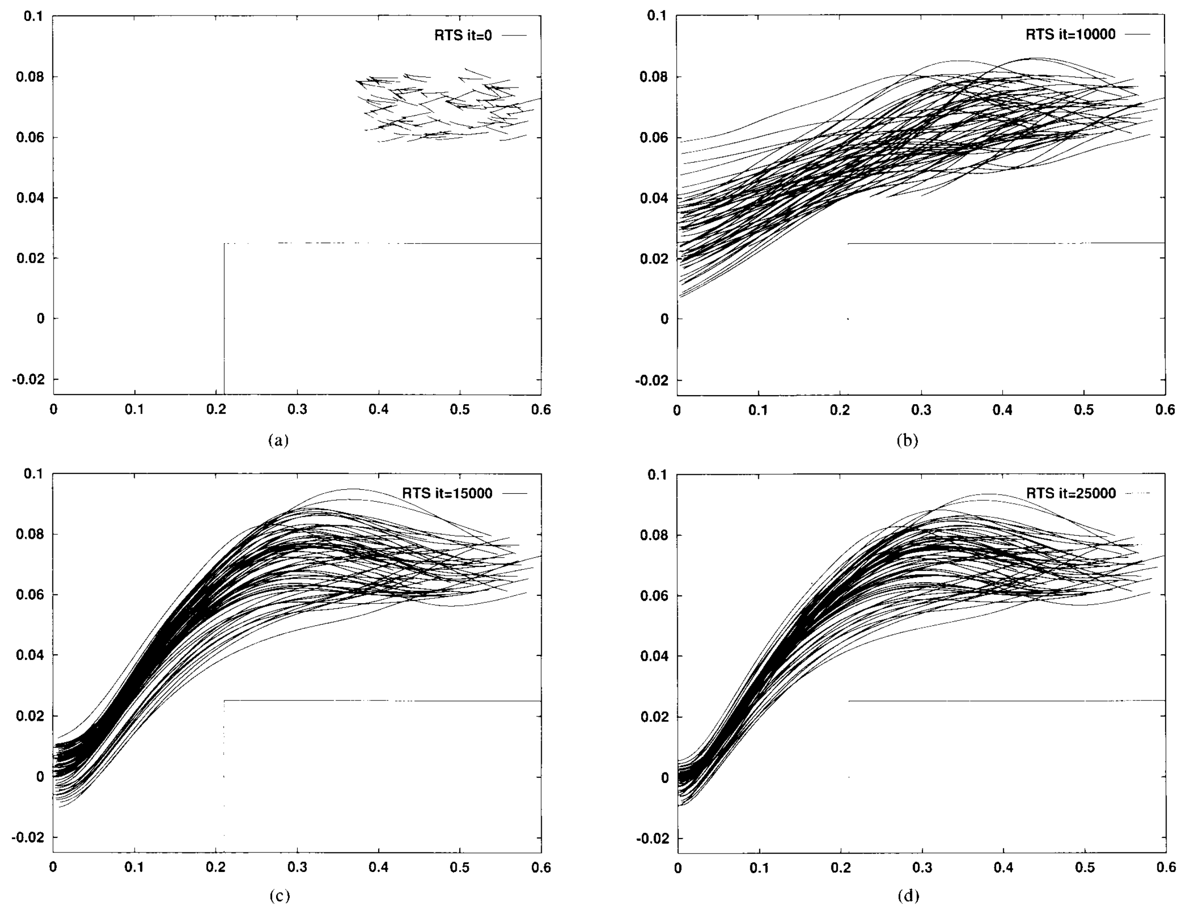


Fig. 12. Evolution of 64 trajectories during training (truck and trailerparking in constrained space) at 0, 10 000, 15 000, and 25 000 iterations.

dynamical system described in Section II. While RTS is based on a single trajectory, in genetic algorithms a population of candidate solutions is considered. The bias toward high-fitness points is obtained by the mutation and selection mechanisms, while the crossover operators build new candidate solutions from the selected individuals. A similar approach based on the combination of multiple solutions (with different combining operators) has been reviewed in [21], with the term scatter search. Recently, the usefulness of a direct "greedy search" component in GA has been recognized in [32]. Detailed comparisons are presented in [9] and [10].

VI. Summary and Conclusion

The present approach goes in opposite direction to a popular approach in the neural network literature: transform a combinatorial optimization problem into a continuous-valued neural net execution [24]. Here an advanced CO technique was used for training neural nets. The heuristic RTS scheme is an effective alternative or a complement to traditional training techniques for solving classification and control problems. In particular, RTS escapes rapidly from local minimizers (Section IV-A), it can approximately duplicate results obtained with BP, and it can obtain better generalization results (Section IV-B).

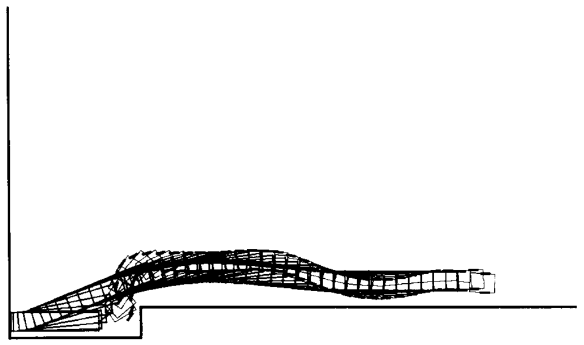


Fig. 13. Truck and trailer motion for a parking task when the system has been trained with RTS, for readability only a subset of the steps is illustrated.

Finally, a nonlinear control task has been solved successfully (Section IV-C).

Let us note that, because training tasks have vastly different characteristics, it is far from our intention to claim that the RTS scheme is the preferred method in all cases. In particular, if the methods based on partial derivatives (like BP) reach satisfactory generalization performances, they can be the fastest techniques to implement on general-purpose computers. A fair comparison of the RTS scheme with BP is difficult because of the different context (initialization, limited precision in the weights and possibly in the squashing function). While it can be the case that BP is appropriate if local minima are not a problem and if the initialization is suitable, the main advantages of the RTS approach are the direct and effective way to continue the search beyond local minima and its extreme flexibility. In particular it is applicable to nondifferentiable (or even discontinuous) performance and transfer functions, and it can easily accommodate weights with a selected number of bits and constraints in the search space (the constraints can be implicit in the encoding or explicit, by limiting the number of available moves). If a special-purpose VLSI circuit is developed for an application, the possibility of realizing a net with a limited number of bits per weight can be cost-effective [37]. Because the steepest descent and the RTS search processes have qualitatively different dynamics, if the generalization results are comparable one can use the networks trained with the two methods in "team" classifiers to limit the bias caused by a single training technique and increase the global performance [4].

Note that the RTS technique is in principle applicable to different network models, both with and without feedback, and to a wide range of machine learning tasks. The competitive advantage of RTS is related to its use of memory (the process is not Markovian): the transitions from a state depend on the past (recent) history of the search. The avoidance of cycles and confinement assures that the available CPU time is spent in an efficient exploration of the search space.

Some open problems are the possibility of evaluating the function on a randomly chosen subset of the training patterns [31], in particular the possibility of passing from a batch to an on-line approach, the study of different sets of basic movements and of schemes with varying resolutions: the discretization can be finer near zero. Because of the lack of derivative computations and the limited precision required, the RTS scheme can be of interest for special-purpose hardware realizations with simple but fast electronic components (see [33] and [25] for some algorithms that are designed by taking the constraints of VLSI realizations into account). In particular, the recently developed TOTEM chip is trained with RTS [11]. Finally, a promising possibility is that of combining RTS search and gradient descent. In a hybrid scheme TABU search can operate with a coarse discretization and gradient descent can be used to reach a high precision in the final result.

Appendix
Worked Out Example of Tabu Search

Let us assume that the search space F is the set of three-bit strings ($f = [b_1, b_2, b_3]$) and the cost function is

$$E([b_1, b_2, b_3]) = b_1 + 2b_2 + 3b_3 - 7b_1b_2b_3.$$

The feasible points (the edges of the three-dimensional binary cube) are illustrated in Fig. 14 with the associated cost function. The neighborhood of a point is the set of points that are connected with edges.

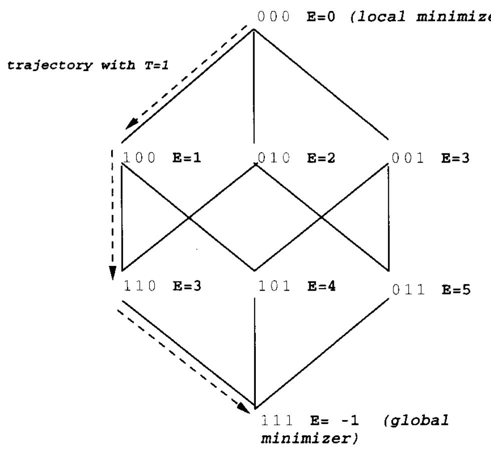


Fig. 14. Feasible points, E values and Tabu trajectory.

The point $f^{(0)} = [0,0,0]$ with $E(f^{(0)}) = 0$ is a local minimizer because all moves produce a higher cost value. The best of the three admissible moves is μ_1, so that $f^{(1)} = [1,0,0]$. Note that the move is applied even if $E(f^{(1)}) = 1 \geq E(f^{(0)})$, so that the system abandons the local minimizer.

If $T^{(1)} = 0$, the best move from $f^{(1)}$ will again be μ_1 and the system will return to its starting point: $f^{(1)} = f^{(0)}$. If $T^{(t)}$ remains equal to zero the system is trapped forever in the limit cycle $[0,0,0] \to [1,0,0] \to [0,0,0] \to [1,0,0] \cdots$.

On the contrary, if $T^{(t)} = 1$, μ_1 is prohibited at $t = 1$ because it was used too recently, i.e., its most recent usage time $\Lambda(\mu_1)$ satisfies $\Lambda(\mu_1) = 0 \geq (t - T^{(t)}) = 0$. The neighborhood is therefore limited to the points that can be reached by applying μ_2 or μ_3 ($N([1,0,0]) = \{[1,1,0],[1,0,1]\}$). The best admissible move is μ_2, so that $f^{(2)} = [1,1,0]$ with $E(f^{(2)}) = 3$.

At $t = 2$ μ_2 is prohibited, μ_1 is admissible again because $\Lambda(\mu_1) = 0 < (t - T^{(t)}) = 1$, and μ_3 is admissible because it was never used. The best move is μ_3 and the system reaches the global minimizer: $f^{(3)} = [1,1,1]$ with $E(f^{(3)}) = -1$.

Acknowledgment

The authors wish to thank Dr. B. Caprile and Dr. C. Furlanello for useful comments on a first draft of the manuscript, and S. Struthers for her kind assistance. Prof. R. Odorico made available the data for the high energy physics discrimination task.

References

[1] C. W. Anderson and W. T. Miller III, "Challenging control problems," in *Neural Networks for Control*, W. T. Miller, III, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: MIT Press, 1991, pp. 575–491.
[2] E. Amaldi, Travail de Diplôme, Département de Mathématiques, Ecole Polytechnique Fédéral de Lausanne, 1988.
[3] R. Battiti, "First- and second-order methods for learning: Between steepest descent and Newton's method," *Neural Computa.*, vol. 4, no. 2, pp. 141-166, 1992.
[4] R. Battiti and A. Colla, "Democracy in neural nets: Voting schemes for classification," *Neural Networks*, vol. 7, no. 4, pp. 691–707, 1994.
[5] R. Battiti and G. Tecchiolli, "Parallel biased search for combinatorial optimization: Genetic algorithms and TABU," *Microprocessors Microsyst.*, vol. 16, no. 7, pp. 351-367, 1992.

[6] ——, "The reactive tabu search," *ORSA J. Comput.*, vol. 6, no. 2, pp. 126–140, 1994.

[7] ——, "Learning with first, second, and no derivatives: A case study in high energy physics," *Neurocomput.*, vol. 6, pp. 181–206, 1994.

[8] R. Battiti, "The reactive tabu search for machine learning," in *Proc. GAA '93, Giornate dei Gruppi di Lavoro AI*IA, Apprendimento Automatico* Milano, Italy, June 1993, pp. 41–55.

[9] R. Battiti and G. Tecchiolli, "Simulated annealing and tabu search in the long run: A comparison on QAP tasks," *Comput. Math. Applicat.*, vol. 28, no. 6, 1994, pp. 1–8.

[10] ——, "Local search with memory: Benchmarking RTS," *OR Spectrum*, in press.

[11] R. Battiti, P. Lee, A. Sartori, and G. Tecchiolli, "TOTEM: A digital processor for neural networks and reactive tabu search," in *Proc. MICRONEURO '94*, Torino, Italy, Sep. 1994, pp. 17–25.

[12] D. A. Beyer and R. G. Ogier, "Tabu learning: A neural network search method for solving nonconvex optimization problems," in *Proc. Int. Joint Conf. Neural Networks* Singapore, Nov. 1991.

[13] E. K. Blum, "Approximation of boolean functions by sigmoidal networks: Part I: XOR and other two-variable functions," *Neural Networks*, vol. 1, pp. 532–540, 1989.

[14] A. L. Blum and R. L Rivest, "Training a 3-node neural network is NP-complete," *Neural Networks*, vol. 5, no. 1, pp. 117–128, 1992.

[15] R. Brunelli and G. Tecchiolli, "Stochastic minimization with adaptive memory," *J. Computa. Appl. Math.*, in press, 1994.

[16] D. F. Elliot and K. R. Rao, *Fast Transforms, Algorithms, Analyses Applications*. Orlando, FL: Academic, 1982.

[17] J. Engel, "Teaching feedforward neural networks by simulating annealing," *Complex Syst.*, vol. 2, pp. 641–648, 1988.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freeman, 1979.

[19] A. H. Gee and R. W. Prager, "Polyhedral combinatorics and neural networks," Cambridge Univ, Eng. Dep., UK, tech. rep. CUED/F-INFENG/TR 100, May 1992.

[20] F. Glover, "Tabu search—Part I," *ORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, 1989.

[21] ——, "Scatter search and star-paths: Beyond the genetic metaphor," Univ. Colorado-Boulder, tech. rep. CO 80309-0419, Mar. 1993.

[22] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[23] M. Gori and A. Tesi, "On the problem of local minima in backpropagation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 14, no. 1, pp. 76–86, 1992.

[24] J. J. Hopfield and D. W. Tank, " "Neural" computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, pp. 141–152, 1985.

[25] M. Jabri and B. Flower, "Weight perturbation: An optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks," *Neural Computa.*, vol. 3, pp. 546–565, 1991.

[26] R. E. Jenkins and B. P. Yuhas, "A simplified neural network solution through problem decomposition: The case of the truck backer-upper," *Neural Computa.*, vol. 4, pp. 647–649, 1992.

[27] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Sci.*, vol. 220, pp. 671–680, 1983.

[28] J. F. Kolen and J. B. Pollack, "Backpropagation is sensitive to initial conditions," *Complex Syst.*, vol. 3, no. 4, pp. 269–280, 1990.

[29] S. Kong and B. Kosko, "Adaptive fuzzy systems for backing up a truck-and-trailer," *IEEE Trans. Neural Networks*, vol. 3, pp. 211–223, 1992.

[30] Z. Luo, "On the convergence of the LMS algorithm with adaptive learning rate," *Neural Computa.*, vol. 3, pp. 227–245, 1991.

[31] M. Møller, "Supervised learning on large redundant training sets," in S. Y. Kung, F. Fallside, J. A. Sorenson. and C. A. Kamm, Eds., *Proc. 1992 IEEE-SP Workshop, Neural Networks Signal Process. II,* Piscataway, NJ, pp. 79–89.

[32] H. Mühlenbein, "The parallel genetic algorithm as function optimizer," *Parallel Comput.*, vol. 3, pp. 619–632, 1991.

[33] A. F. Murray, "Multilayer perceptron learning optimized for on-chip implementation: A noise-robust system," *Neural Computa.*, vol. 4, pp. 366–381, 1992.

[34] D. H. Nguyen and B. Widrow, "The truck backer-upper: An example of self-learning in neural networks," in *Proc. Int. Joint Conf. Neural Networks (IJCNN-89)*, Washington, DC, vol. II, June 1989, pp. 357–363.

[35] P. Mazzanti and R. Odorico, "Bottom jets recognition by neural networks and statistical discriminants: A survey," Univ. Bologna, Italy, dept. Physics rep. DFUB 92/15, Zeitschrift für Physik C, in press, 1992.

[36] R. Odorico, "COJETS 6.23: A Monte Carlo simulation program for antiproton-proton, Proton-proton collisions and electron-positron annihilation," *Comput. Phys. Commun.*, vol. 72, pp. 238–248, 1992.

[37] M. Maresi, G. Orlandi, F. Piazza and A. Uncini, "Fast neural networks without multipliers," *IEEE Trans. Neural Networks*, vol. 4, no. 1, pp. 53–62, 1993.

[38] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Englewood Cliffs, NJ: Prentice-Hall, 1982.

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*. Cambridge, MA: MIT Press, vol. 1, 1986, pp. 318–362.

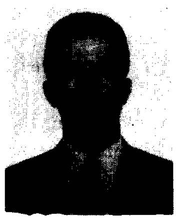
[40] S. Saarinen, R. Bramley, and G. Cybenko, "The numerical solution of neural network training problems," *SIAM J. Statistical Scientific Comput.*, in press, 1993.

[41] F. J. Solis and R. J.-B. Wets, "Minimization by random search techniques," *Math. Operations Res.*, vol. 6, no. 1, pp. 19–30, 1981.

[42] D. de Werra and H. Hertz, "Tabu search techniques: A tutorial and an application to neural networks," *OR Spectrum*, vol. 11, pp. 131–141, 1989.

[43] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computa.*, vol. 1, pp. 270–280, 1989.

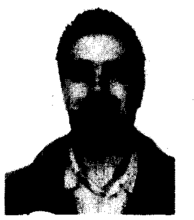
[44] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Compu.*, vol. 14, pp. 347–361, 1990.



Roberto Battiti received the bachelor's degree in 1985 from Trento University, Italy, and the Ph.D. degree in computation and neural systems from California Institute of Technology, Pasadena, 1990.

He is a Faculty Member at the Department of Mathematics of the University of Trento, Italy, and an Associate of the Istituto Nazionale di Fisica Nucleare (INFN). He has been a Consultant for the industrial application of neural networks and concurrent processing and he is a Participant of the special INFN initiative for realizing RTS-specific VLSI chips to be used in high energy physics detectors and signal processing applications. His main research interests are machine learning techniques, combinatorial and continuous optimization algorithms, the design and analysis of algorithms for massively parallel architectures, and special purpose VLSI circuits.

Dr. Battiti is a member of the IEEE Computer Society.



Giampietro Tecchiolli was born in Trento, Italy in 1961. He received the bachelor's degree in physics from the University of Trento in 1986.

Since 1987 he has been with INFN working in the area of parallel and distributed computing for computational theoretical physics. He is a Participant of the special INFN initiative for realizing RTS-specific VLSI chips and co-designer of the TOTEM chip. He joined IRST in 1988, where he works in the System Architectures Laboratory. Since 1993 he has been with the Computer Science Departement at the University of Verona where he is Adjunct Professor. His research interests include discrete, continous and functional optimization, parallel and distributed computation, computer architectures, and algorithm complexity.