Learning with First, Second, and No Derivatives: a Case Study in High Energy Physics

Roberto Battiti

Dipartimento di Matematica, Università di Trento, 38050 Povo (Trento) - Italy

and INFN, Gruppo Collegato di Trento

e-mail: battiti@itnvax.cineca.it

GIAMPIETRO TECCHIOLLI

Istituto per la Ricerca Scientifica e Tecnologica 38050 Povo (Trento) - Italy

and INFN, Gruppo Collegato di Trento

e-mail: tec@irst.it

February 15, 1993

Neurocomputing, in press

In this paper different algorithms for training multi-layer perceptron architecture are applied to a significant discrimination task in High Energy Physics. The One-Step Secant technique is compared with On-Line Backpropagation, the "Bold Driver" batch version and Conjugate Gradient methods. In addition, a new algorithm (Affine Shaker) is proposed that uses stochastic search based on function values and affine transformations of the local search region. Although the Affine Shaker requires more CPU time to reach the maximum generalization, the technique can be interesting for special-purpose VLSI implementations and for non-differentiable functions.

1 Introduction

Now that the sub-symbolic paradigm for the development of systems with cognitive capabilities is recognized both in the "cognitive science" and in the "artificial intelligence" community, it is worth considering briefly the historical development. When cybernetics began to study machines capable of cognitive behaviors, the approach was strongly operational, aiming at achieving high-level functions by starting from basic mechanisms (see [30] for a historical overview). In particular, a strong emphasis was put on the self-organization properties: the adaptation and learning capabilities of these systems were explained as the gradual optimization of functions describing the desired functionality. The dynamical systems approach to intelligence dates back at least to Caianiello [11] and Amari [1], to cite just two pioneers. The current renaissance is in part related to the actual calculation of the partial derivatives needed with effective algorithms (like the well known error-backpropagation of [27]), to the cultural interest for non-linear systems, to the availability of fast sequential and parallel processors and, last but not least, to the development of applications using simple forms of *gradient descent* that were shown to be remarkably effective. The use of gradual adaptations with simple and local mechanisms permits a close link with neuroscience, a continuing source of models and inspiration for the neural network research community, although the detailed realization of gradient descent algorithms with real neurons is still a research subject.

As soon as the importance of optimization for learning was recognized, researchers began to use techniques derived from the optimization literature that use higher-order derivative information during the search, going *beyond gradient descent.* Examples are the conjugate gradient and the "secant" methods, i.e., methods that update an approximation of the Hessian in an iterative way by using only gradient information. In fact, it is well known that taking the gradient as the current search direction produces very slow convergence speed if the Hessian has a large *condition number* (in a pictorial way this corresponds to having "narrow valleys" in the search space). Techniques based on second-order information are in widespread use in the neural net community, their utility being recognized in particular for problems with a limited number of weights (< 100) and requiring high precision in the output values. A partial bibliography and a description of the relationships between different second-order techniques was presented in [3].

In spite of their utility, the above techniques are not completely satisfactory. In particular their realization with analog VLSI hardware is problematic mainly because of the high precision required. In addition, many tasks are characterized by high degrees of hill-conditioning and they are therefore prone to numerical errors and inefficiencies. If the use of first and second derivatives has difficulties, one is motivated to study techniques that do not need derivatives, like methods derived from combinatorial optimization (see the ***Reactive Tabu*** search in [6]) or from stochastic search (see [12] and [7]). In this context the link with neuroscience becomes very weak or completely absent, but the techniques can be competitive for the development of artificial cognitive systems built with VLSI components, where the design is subject to different constraints.

The purpose of this paper is that of benchmarking "traditional" optimization techniques and a new method based only on function values (obtained with a smart sampling of the neighborhood) applied to a significant discrimination task in the area of High Energy Physics. The paper is therefore divided into two parts: the first discuss training and test results obtained with a selection of the methods reviewed in [3], the second presents the new optimization algorithm and the results obtained on the same task.

The benchmark task is illustrated in Section 2. The methods based on backpropagation with modifications and the results obtained are described in Section 3, while the algorithms based on "blind" stochastic search are presented in Section 4. Finally a brief comparison of the different schemes is discussed in Section 5.

2 The Task: Event Discrimination in High Energy Physics

High Energy Physicists must address challenging pattern recognition problems that arise in experiments based on collider facilities. In a typical setup, the colliding particles produce streams of secondary particles ("jets") that leave traces on a set of detectors that measure trajectory parameters and energy deposited (the energy detectors are called "calorimeters"), see, for example, [17].

During the last years a growing number of contributions have appeared on the use of neural networks as pattern classifiers in High Energy Physics. In particular neural nets have been proposed for discriminating bottom quark jets at LEP, the Large Electron-Positron collider at the CERN laboratories.

In this paper the task of recognizing 2-jet events produced by the bottom quark among a background of non-bottom jets is used as a benchmark for different training algorithms for feed-forward neural networks. For the physics behind the discrimination task and for a recent review of neural network approaches to the problem see [20].

The patterns used for training and testing the neural classifiers were produced with the COJETS event generator [23, 24], using the natural frequencies. At present, the search for the particles produced by the bottom quark is a challenging task for experimental physicists (the quark itself is not directly observable, if the "quark confinement" theoretical hypothesis holds).

A total of 100,000 e^+e^- events are generated at center-of-mass-system energy of 91 Gev. Of these only 2-jet events are retained, and among these only the jets with at least four particles are selected. Each jet is described by a pattern with 17 features, corresponding to the variables described in [20].

The calorimeter granularity is defined by cells of 1 degree $\times$ 1 degree. The input variables are obtained from a square of 64×64 cells centered on the jet axis, see Figure 1. In detail, the 17 features are derived by counting cells or groups of them with various threshold requirements on the cell energy deposition E_{cell}, and by summing energy depositions over cells satisfying various thresholds on E_{cell} and angular cuts:

1-5 number of cells with $E_{cell} > 0.5,\ 1,\ 2,\ 4,\ 8$ Gev

6-15 energy depositions in regions R2 and R3 of Figure 1, including only cells with $E_{cell} > 0,\ 1,\ 2,\ 4,\ 8$ Gev

16 energy deposition in the region R1 of Figure 1, including only cells with $E_{cell} >\ 0.5$

17 number of the sixteen 16×16 cell groups having energy deposition $E_{16\times16} > 0.5$ Gev.

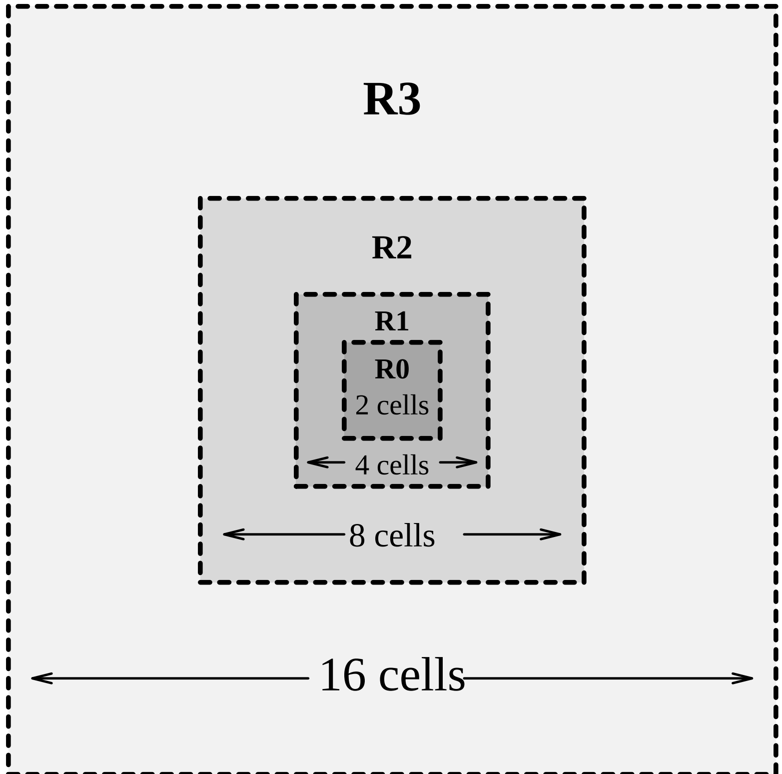


Figure 1: Calorimetric cell grid around the jet axis used to extract the 17 calorimetric jet variables (central region). Each region is identified by a different color.

Note that the central region (R0 in Figure 1) is not used because the information content is non significant.

The jets originated by the bottom quark have been subdivided 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 have been subdivided into two sets of 25,463 events each.

In the proposed application in HEP, one is interested in improving the signal-to-noise ratio. To this end, the threshold for a "bottom" recognition is varied, and the tradeoff between the *purity* of the patterns recognized as bottom (i.e., the recognition accuracy) and the *efficiency* (i.e., the fraction of the original "bottom" jets that are recognized) is measured. The purity-efficiency curve that can be obtained from the features used is illustrated in Figure 2. The two curves have been obtained by using the *One-Step Secant* (OSS) algorithm (see section 3.4) with two different training sets, while the same test set with 7,741 + 25,463 patterns has been used only for deriving

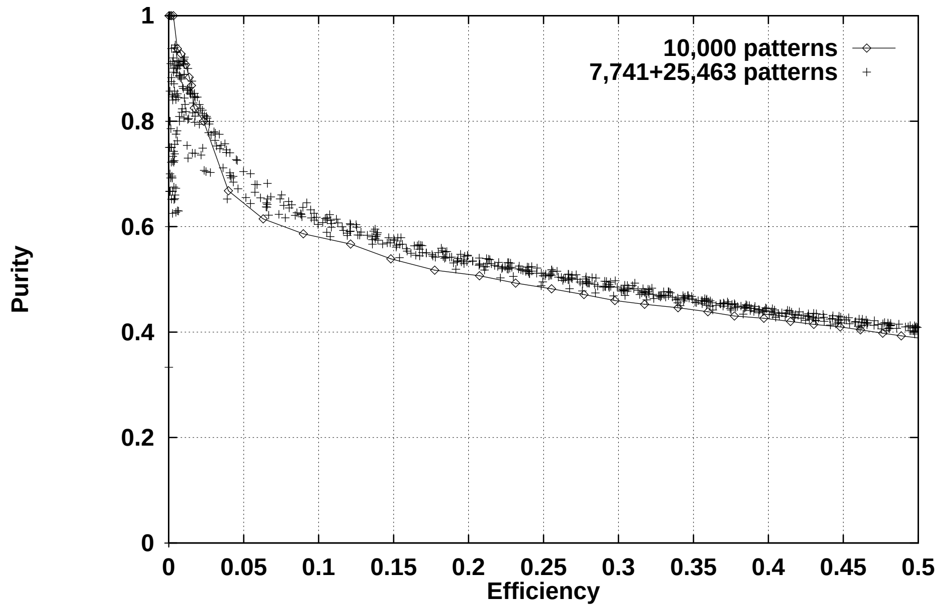


Figure 2: Purity versus efficiency results for bottom jet discrimination using the COJETS event generator

the recognition accuracy. One training set is composed of 5000 "bottom" patterns and of 5000 "background" patterns, the other one with the complete training set of 7,741 + 25,463 patterns.

The results for the larger training set are shown at different phases of the OSS Algorithm (see Section 3.4), between 80 and 160 iterations. The results for the smaller set are shown after 80 iterations. Because the two cases are not significantly different we select as benchmark the set with 10,000 patterns. In detail, 5,000 "bottom" and 5,000 "non-bottom" patterns are extracted from the data files to build a training set and a disjoint test set (with 10,000 patterns each). The network architecture has a single hidden layer with 10 units and a single output unit (target is 1 for "bottom", 0 for "non bottom"). 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'. Separating the two classes is hard, in fact high-purity results are obtained only at the price of low efficiencies (see Figure 2). The low generalization accuracy of all tests presented in this paper is a second indication that the Bayesian limit (the best accuracy obtainable given the two class probability distributions) is probably low.

3 Methods Using First and Second Derivatives

Let us briefly define the notation. We consider the "standard" multi-layer perceptron architecture, with weights connecting only nearby layers and the sum-of-squared-differences *energy* function defined as:

$$E(w) = \frac{1}{2}\sum_{p=1}^{P} E_p = \frac{1}{2}\sum_{p=1}^{P}(t_p - o_p(w))^2 \tag{1}$$

where t_p and o_p are the target and the current output values for pattern p, respectively. The sigmoidal transfer function is $f(x) = 1/(1+e^{-x})$.

The initialization is the same for all tests: the initial weights are randomly distributed in the range $(-.5, .5)$.

In the following sections we present the experimental results obtained with two "gradient-based" techniques: the on-line backpropagation of [27], and a version of *batch* backpropagation with adaptive learning rate ("bold-driver" BP, see [4]), and two techniques that use second-derivatives information (in an indirect and fast way): the conjugate gradient technique and the one-step-secant method with fast line searches (OSS, see [3], [4]).

The algorithms considered have been selected in order to test some significant and radically different approaches to training the multilayer perceptron architecture and the set is clearly not exhaustive.

3.1 On-Line BP

The stochastic on-line back-propagation update is given by:

$$w_{k+1} = w_k - \epsilon \nabla E_p(w_k) \tag{2}$$

where the pattern p is chosen randomly from the training set at each iteration and ϵ is a fixed learning rate (in this test the use of momentum is not considerd, but see Section 3.3). This form of backpropagation has been used with success in many contexts, provided that an appropriate learning rate is selected by the user. The main difficulties of the method are that the iterative procedure is not guaranteed to converge and that the use of the gradient as a search direction is very inefficient for ill-conditioned problems. The competitive advantage with respect to *batch* backpropagation, where the complete gradient of E is used as a search direction, is that the partial gradient $\nabla E_p(w_k)$ requires only a single forward and backward pass, so that the inaccuracies of the method can be compensated by the low computation required by a single iteration, especially if the training set is large and composed of redundant patterns. In this case, if the learning rate is appropriate, the actual CPU time for convergence can be small.

In Figures 3 and 4 we show the test results of the technique for two significant values of the learning rate that are at the boundaries of the interesting range ($\epsilon_1 = 10^{-3}$ and $\epsilon_2 = 10^{-1}$). To avoid cluttering the graph, we present only the generalization results, for four typical experiments out of ten. The average squared error (or the energy in equation 1 multiplied by 2 and divided by the number of patterns) and the percent accuracy of recognition is measured at various points during the training process until the over-training phase begins, and the generalization results worsen.

It can be observed that the smaller learning rate is suitable for the current task: if ϵ decreases below ϵ_1 the training time increases without producing better generalization results, while if ϵ grows beyond ϵ_2 the oscillations become gradually wilder, and the uncertainty in the generalization obtained increases, as can be seen from the larger amplitude of the oscillations of the accuracy values.

Let us note that the best generalization results are obtained for "large" E values (and this result is confirmed in the following tests), just a reminder of the fact that converging to very small E values is not required in some discrimination tasks. The situation is therefore different than the typical *optimization* context. In particular the assumption of *small residuals* often fails completely.

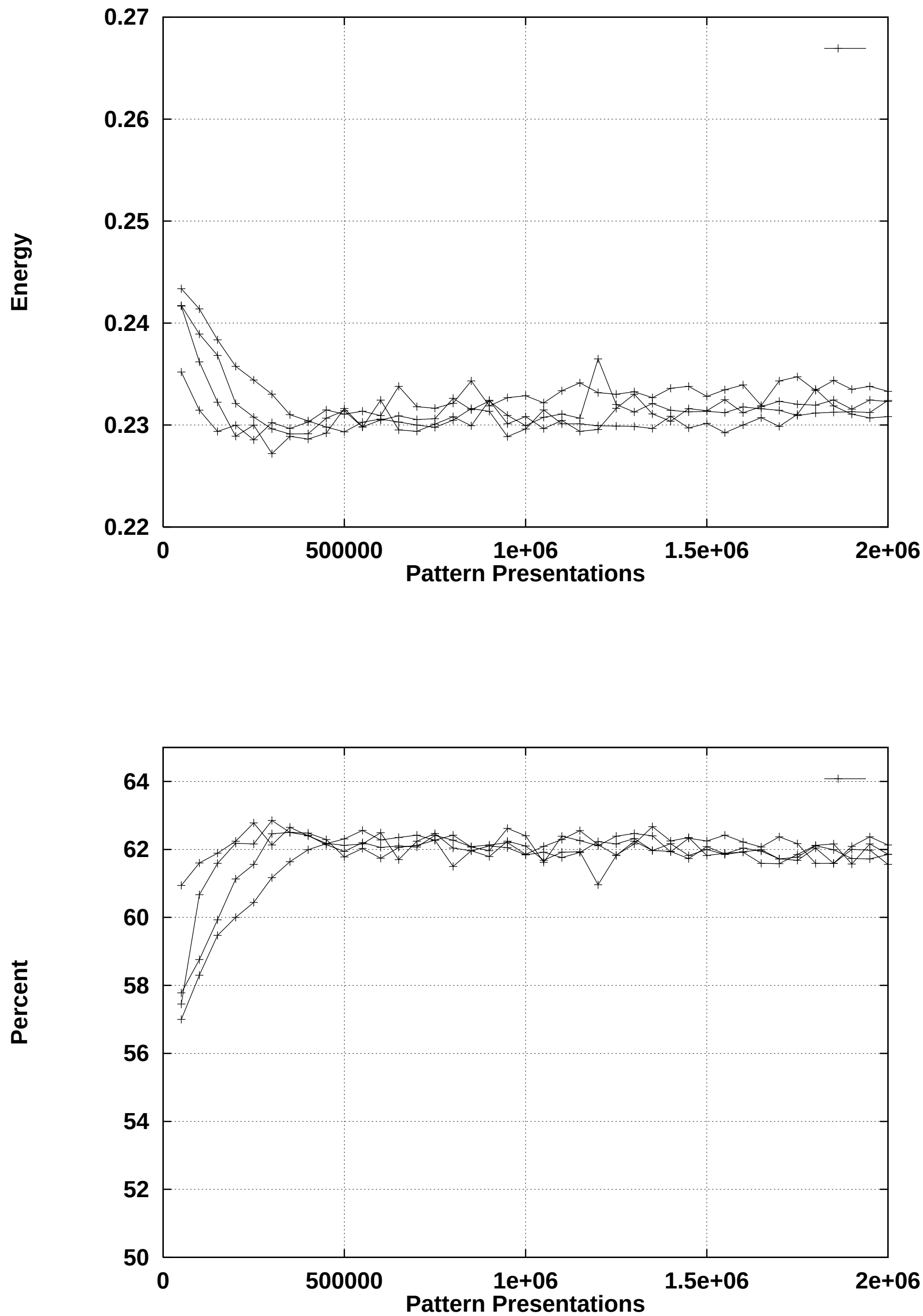


Figure 3: Generalization results for the On-Line BP algorithm with learning rate = 10^{-3}. Average squared error (above) and recognition accuracy (below)

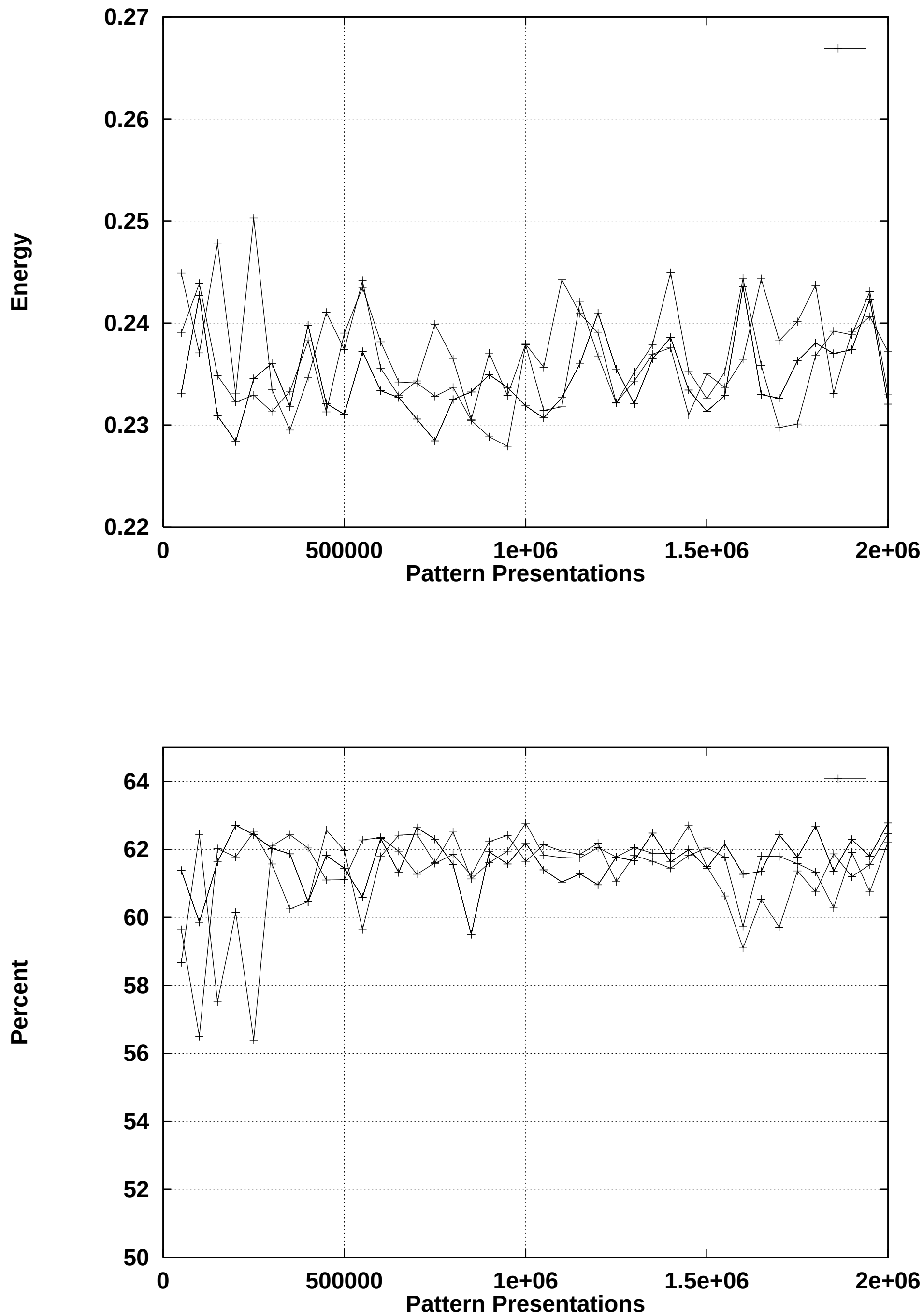


Figure 4: Generalization results for the On-Line BP algorithm with learning rate = 10^{-1}. Average squared error (above) and recognition accuracy (below)

3.2 Batch "Bold-Driver" BP

The batch back-propagation update is a form of gradient descent defined as:

$$w_{k+1} = w_k \; - \; \epsilon \; g_k \tag{3}$$

where g_k is the gradient of the ***energy*** function: $g_k = \nabla E(w_k)$.

The previous update, with a fixed ***learning rate*** ϵ, can be considered as a crude version of ***steepest descent***, where the exact minimum along the gradient direction is searched at each iteration:

$$w_{k+1} = w_k - \epsilon_k g_k \tag{4}$$

$$\text{where} \quad \epsilon_k \;\; \text{minimizes} \quad E(w_k - \epsilon g_k) \tag{5}$$

$$\tag{6}$$

A heuristic proposal for modifying the learning rate is the ***bold driver*** (BD) method described in [4]. The learning rate increases exponentially if successive steps reduce the energy, and decreases rapidly if an "accident" is encountered (if E increases), until a suitable value is found. After starting with a small learning rate, its modifications are described by the following equation:

$$\epsilon(t) = \begin{cases} \rho \; \epsilon(t-1) & \text{if } E(w(t)) < E(w(t-1)) \\ \sigma^l \; \epsilon(t-1) & \text{if } E(w(t)) > E(w(t-1)) \text{ using } \epsilon(t-1) \end{cases} \tag{7}$$

where ρ is close to one ($\rho = 1.1$) in order to avoid frequent "accidents" because the energy computation is wasted in these cases, σ is chosen to provide a rapid reduction ($\sigma = 0.5$), and l is the minimum integer such that the reduced rate $\sigma^l \epsilon(t-1)$ succeeds in diminishing the energy. Similar heuristics have been used in [19] and [32].

The performance of this "quick and dirty" form of backpropagation is close and usually better than the one obtained by appropriately choosing a ***fixed*** learning rate. Nonetheless, being a simple form of ***steepest descent***, the technique suffers from the common limitation of techniques that use the gradient as a search direction. In fact, it can be shown that, when steepest descent is used to minimize a quadratic function $Q(w) = c^T w + \frac{1}{2} w^T G w$ (G symmetric and positive definite) the convergence can become very slow. In detail:

$$|Q(w_{k+1}) - Q(w_*)| \approx \left(\frac{\eta_{max} - \eta_{min}}{\eta_{max} + \eta_{min}}\right)^2 |Q(w_k) - Q(w_*)| \tag{8}$$

where η_{max} and η_{min} are the maximum and minimum eigenvalues of the matrix, and w_* is the minimizer. If the two extreme eigenvalues are very different, the distance from the minimum value is multiplied each time by a number that is close to one.

The ratio between maximum and minimum eigenvalue is related to the ***condition number.*** In detail, the condition number $\kappa(H)$ of a matrix H is defined as $||H||\,||H^{-1}||$, where $||\ ||$ is the matrix operator norm induced by the vector norm: $\|H\| = \max_x(\|Hx\|/\|x\|)$. The conditioning number is the ratio of the maximum to the minimum stretch induced by H and measures, among other effects, the sensitivity of the solution of a linear system to finite-precision arithmetic. If a linear system $H\ x\ =\ b$ is perturbed in the following way:

$$(H + \epsilon F)x(\epsilon) = b + \epsilon f \tag{9}$$

the relative error in the solution can be bounded as:

$$\frac{\|x(\epsilon) - x\|}{\|x\|} \leq \kappa(H)\,(\frac{\|\epsilon F\|}{\|H\|} + \frac{\|\epsilon f\|}{\|b\|}) + O(\epsilon^2) \tag{10}$$

Thus, the relative error in x can be $\kappa(H)$ times the relative error in H and b (see [16] for the details).

The condition number can be obtained with the singular value decomposition (SVD) of the Hessian matrix. In this decomposition the Hessian matrix is decomposed as:

$$\mathbf{H} = \mathbf{U}\mathbf{S}\mathbf{V}^T = \sum_{i=1}^{N} s_i u_i v_i^T \tag{11}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices and $\mathbf{S}$ is diagonal $\mathbf{S} = \mathrm{diag}(s_i)$. The condition number is given by the ratio of the largest of the s_i to the smallest of the s_i. It can be shown that, in the case of a symmetric matrix, the singular values are the absolute values of the corresponding eigenvalues of the matrix.

In Newton's method for optimization the step s at a given iteration minimizes the *local model* given by the first three terms of the Taylor series expansion of the function to be optimized. s is obtained by solving the linear system:

$$H\ s\ =\ -g \tag{12}$$

Using the SVD of equation 11:

$$s\ =\ -H^{-1}\ g = -(\mathbf{U}\mathbf{S}\mathbf{V}^T)^{-1}\ g = -\sum_{i=1}^{N} \frac{u_i^T g}{s_i} v_i \tag{13}$$

it is apparent that small changes in H or g cause large changes in s if some singular values s_i are small.

A matrix is seriously *ill-conditioned* if the reciprocal of its condition number is too large, or if it approaches the machine's floating point precision (about 10^{-6} for single precision or 10^{-12} for double). Using the exact calculation of the Hessian matrix described in [8] and the SVD routines of [26], we monitored the *ill-conditioning* of the Hessian during training (in the `svdcmp` routine of [26] double precision computations have been used and the maximum number of iterations has been increased from 30 to 300).

As can be observed from the training curves of Figure 5, the condition number is already large at the beginning and becomes much larger as training progresses. Larger condition numbers are related to a serious slowing down of the rate of convergence caused by the effect described in equation 8 (valid for a quadratic model) and by the inaccuracies of finite precision computation (equation 10). Hessian ill-conditioning is common in feed-forward neural nets and this problem must be taken into consideration when designing suitable optimization algorithms.

A detailed analysis of the reasons why ill-conditioning is common is presented in [29]. According to the authors "formulating feed-forward neural network problems as least squares problems can cause undue strain on any numerical scheme which uses Jacobians and a re-formulation of the problem is called for". The approaches described in Section 4 are a possible step toward sidestepping this problem.

The generalization results of the *bold driver* method are shown in Figure 6.

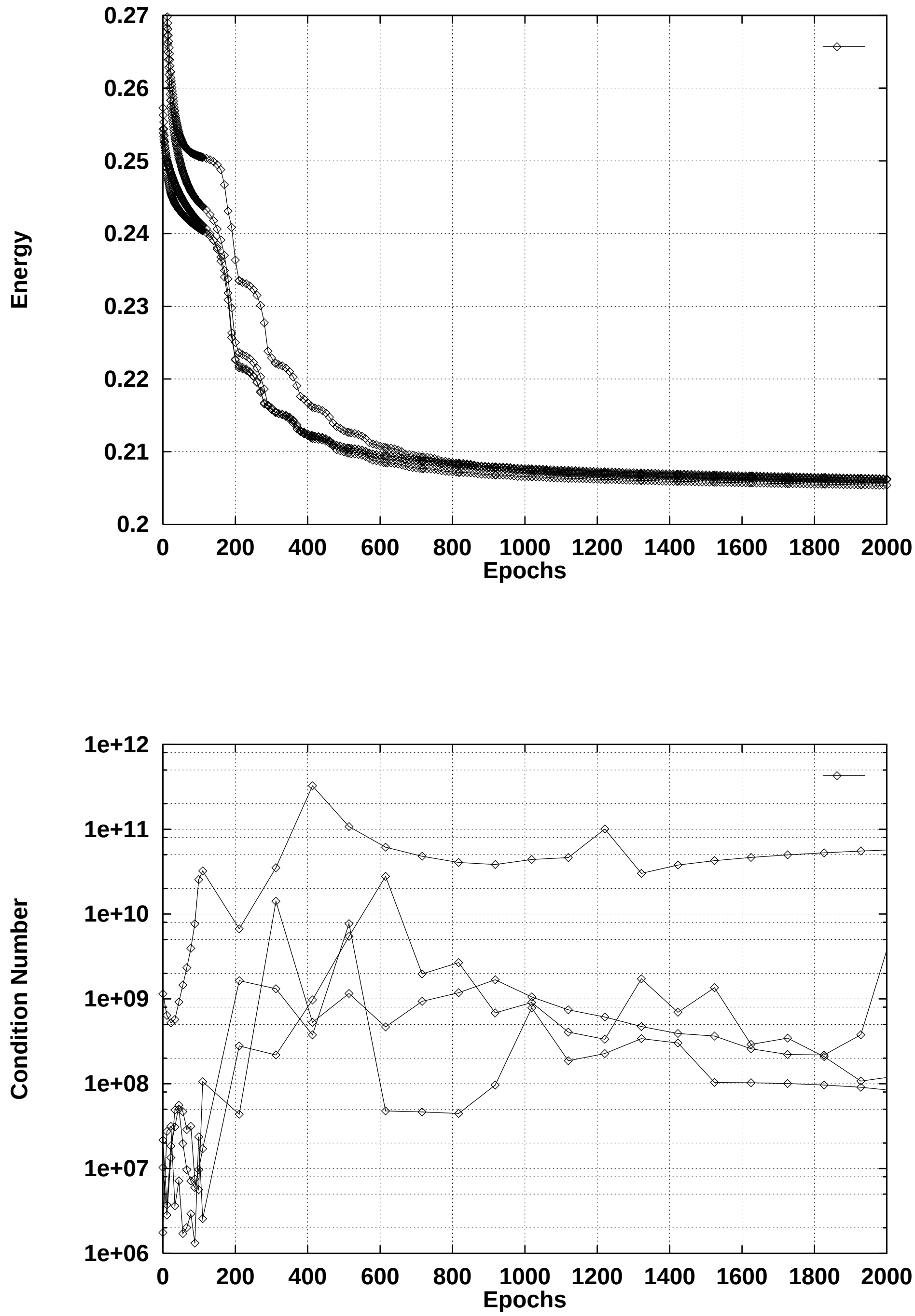


Figure 5: Training curves for the Bold-Driver BP algorithm (above) and corresponding condition number of the exact Hessian matrix (below).

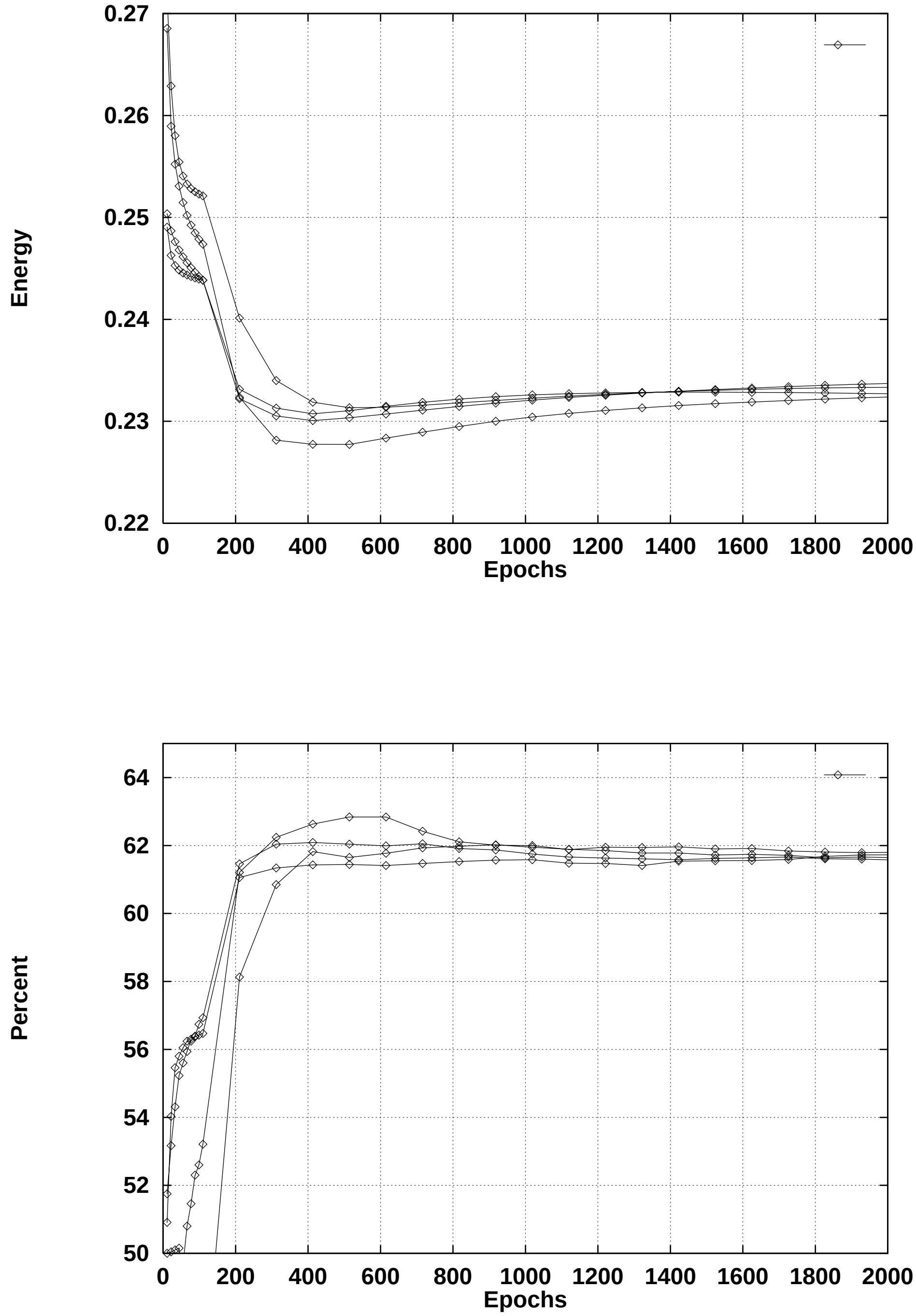


Figure 6: Generalization results for the Bold-Driver BP algorithm. Average squared error (above) and recognition accuracy (below)

3.3 Conjugate Gradient

The concept of ***non-interfering*** directions is at the basis of the conjugate gradient method (CG) for minimization. Two directions are ***mutually conjugate*** with respect to the matrix G if:

$$p_i^T G p_j = 0 \quad \text{when} \quad i \neq j \tag{14}$$

After minimizing in direction p_i, the gradient at the minimizer will be perpendicular to p_i. If a second minimization is in direction p_{i+1}, the change of the gradient along this direction is $g_{i+1} - g_i = \alpha G p_{i+1}$ (for some constant α). The matrix G is indeed the Hessian, the matrix containing the second derivatives, and in the quadratic case the model coincides with the original function. Now, if equation 14 is valid, this change is ***perpendicular*** to the previous direction ($p_i^T(g_{i+1} - g_i) = 0$), therefore the gradient at the new point ***remains*** perpendicular to p_i and the previous minimization is not spoiled. While for a quadratic function the conjugate gradient method is guaranteed to converge to the minimizer in at most $(N+1)$ function and gradient evaluations (at least for infinite-precision calculations), for a general function the steps must be iterated until a suitable approximation to the minimizer is obtained.

Let us introduce the vector $y_k = g_{k+1} - g_k$. The first search direction p_1 is given by the negative gradient $-g_1$. Then the sequence w_k of approximations to the minimizer is defined by:

$$w_{k+1} = w_k + \alpha_k p_k \tag{15}$$

$$p_{k+1} = -g_{k+1} + \beta_k p_k \tag{16}$$

$$\tag{17}$$

where g_k is the gradient, α_k is chosen to minimize E along the search direction p_k and β_k is given either by:

$$\beta_k = \frac{y_k^T g_{k+1}}{g_k^T g_k} \qquad \text{(Polak-Ribiere choice)} \tag{18}$$

or by:

$$\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k} \qquad \text{(Fletcher-Reeves choice)} \tag{19}$$

The different choices coincide for a quadratic function [31]. A major difficulty with the above forms is that, for a general function, the obtained directions are *not* necessarily descent directions and numerical instability can result.

The use of a *momentum* term to avoid oscillations in [27] can be considered as an approximated form of conjugate-gradient. In both cases, the gradient direction is modified with a term that takes the previous direction into account, the important difference being that the parameter β in conjugate-gradient is automatically defined by equations 18 and 19.

Figures 7 and 8 show the generalization results obtained by using a library version of the CG technique described in [26] (routine `frprmn` with tolerance $= 10^{-9}$).

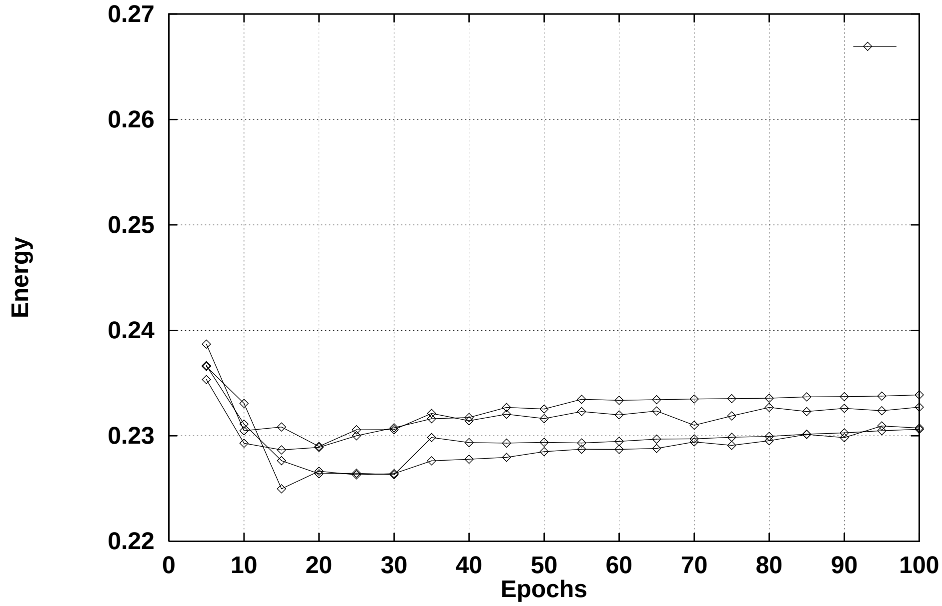
It can be noted that the method requires a very limited number of iterations (about 20) to reach the maximum generalization. Unfortunately, each iteration requires a sizable number of function calls (about 30) and gradient calls (about 25). The two variants do not present statistically significant differences.

Some references about the use of conjugate gradient for training in neural networks are, for example, [18], [21] and [2].

3.4 One-Step Secant with Fast Line Search

Computing the exact Hessian requires order $O(N^2)$ operations [8] and order $O(N^2)$ memory to store the Hessian components, in addition the solution of equation 13 to find the step (or search direction) in Newton's method requires $O(N^3)$ operations, at least when using traditional linear algebra routines. Fortunately, some second-order information can be calculated by starting from the last gradients, and therefore reducing the computation and memory requirements to find the search direction to $O(N)$. The term "secant methods" used in [14] is reminiscent of the fact that derivatives are approximated by the secants through two function values, although in many dimensions the function values (here the function is the gradient) do not determine uniquely the "secant" (here the approximated Hessian).

Historically the one-step-secant method *OSS* is a variation of what is called *one-step (memory-*



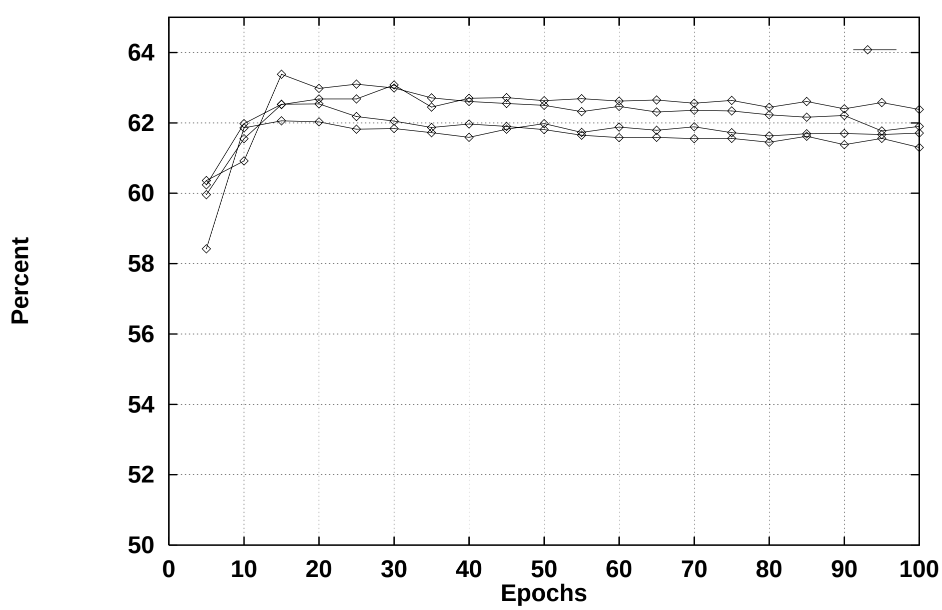
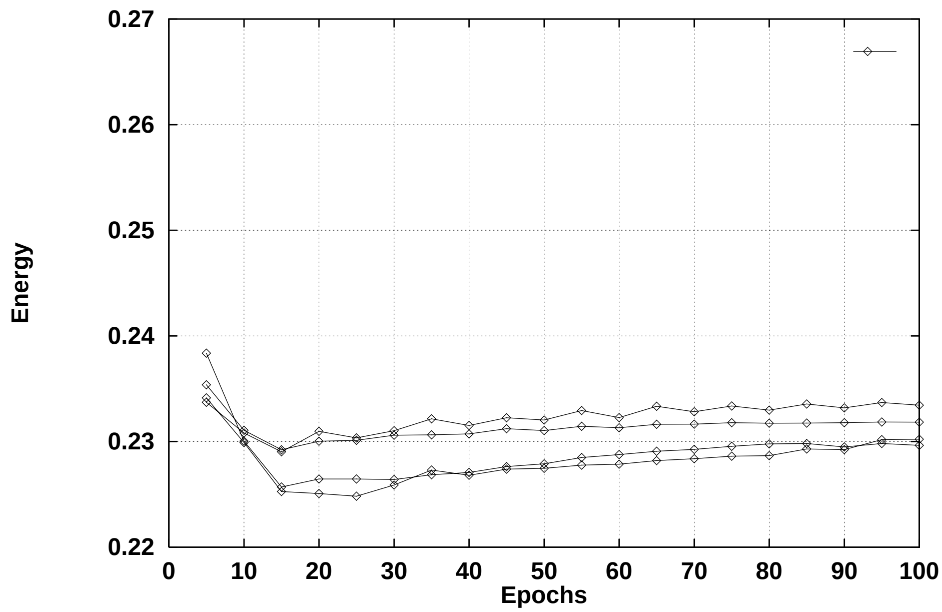


Figure 7: Generalization results for the Conjugate Gradient BP (Polak-Ribiere) algorithm. Average squared error (above) and recognition accuracy (below)



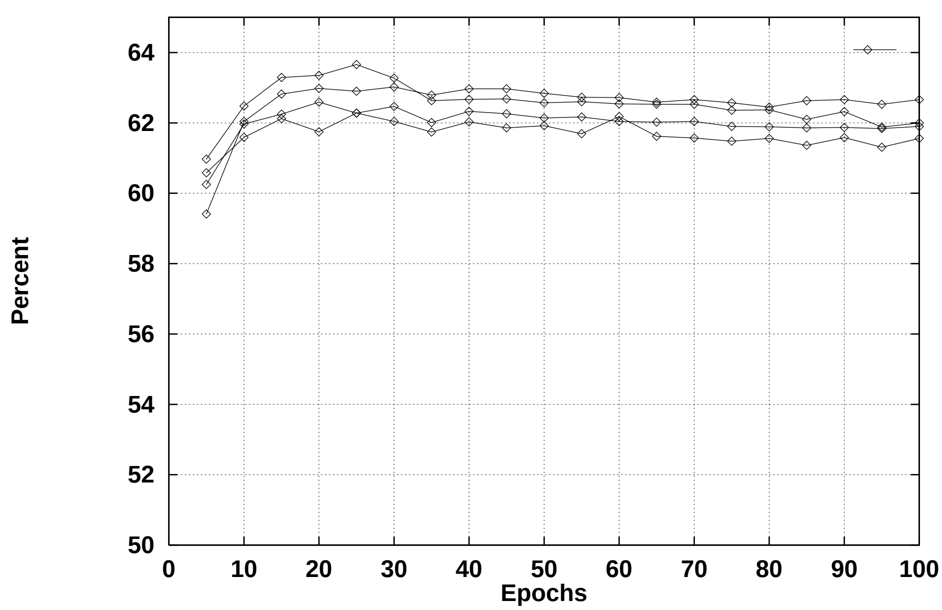


Figure 8: Generalization results for the Conjugate Gradient BP (Fletcher-Reeves) algorithm. Average squared error (above) and recognition accuracy (below)

less) Broyden-Fletcher-Goldfarb-Shanno method, see [31], [13]. The OSS method has been used for multilayer perceptrons in [4] and [5]. The main procedures are illustrated in Figure 9.

Note that BFGS (see [33]) stores the whole approximated Hessian, while the ***one-step*** method requires only vectors computed from gradients. In fact, the new search direction p_{k+1} is obtained as:

$$p_{k+1} = -g_{k+1} + A_k s_k + B_k y_k \tag{20}$$

where the two scalars A_k and B_k are the following combinations of scalar products of the previously defined vectors s_k, g_{k+1} and y_k (last step, current gradient and difference of gradients: $y_k = g_{k+1} - g_k$):

$$A_k = -\left(1 + \frac{y_k^T y_k}{s_k^T y_k}\right) \frac{s_k^T g_{k+1}}{s_k^T y_k} + \frac{y_k^T g_{k+1}}{s_k^T y_k} \quad ; \quad B_k = \frac{s_k^T g_{k+1}}{s_k^T y_k}$$

The search direction is the negative gradient at the beginning of learning and it is restarted to $-g_{k+1}$ every N steps (N being the number of weights in the network).

The fast one-dimensional minimization along the direction p_{k+1} is crucial to obtain an efficient algorithm. This part of the algorithm (derived from [14]) is described in Figure 10. The one-dimensional search is based on the "backtracking" strategy. The last successful learning rate λ is increased ($\lambda \leftarrow \lambda \times 1.1$) and the first tentative step is executed. If the new value E is not below the "upper-limiting" curve, then a new tentative step is tried by using successive quadratic interpolations until the requirement is met. Note that the learning rate is decreased by L_{decr} after each unsuccessful trial. Quadratic interpolation is not wasting computation, in fact, after the first trial one has exactly the information that is needed to fit a parabola: the value of E_0 and E'_0 at the initial point and the value of E_λ at the trial point. The parabola $P(x)$ is:

$$P(x) = E_0 + E'_0 x + \left[\frac{E_\lambda - E_0 - \lambda E'_0}{\lambda^2}\right] x^2 \tag{21}$$

and the minimizer λ_{min} is:

$$\lambda_{min} = \frac{-E'_0}{2\left[\frac{E_\lambda - E_0 - \lambda E'_0}{\lambda^2}\right]} \leq \frac{1}{2(1 - G_{decr})}\lambda \tag{22}$$

```
procedure oss_minimize
begin
begin_or_restart
Initialize the learning rate ε := 10^-5.
Initialize the average learning rate ε̄ := 10^-5.
Set w⃗_curr := random initial weights
iterations := 1
while convergence_criterion_is_not_satisfied do
    begin
    if iterations_is_a_multiple_of_N then
        begin_or_restart
    iterations := iterations + 1
    d⃗ := find_search_direction (see equation 20)
    if fast_line_search (d⃗) = FALSE then
        begin_or_restart
    end
end


procedure begin_or_restart
begin
Find the current energy value.
ε := ε̄
d⃗ := – g⃗
fast_line_search (d⃗)
end
```

Figure 9: Pseudo-code description of the One-Step Secant algorithm. Part I

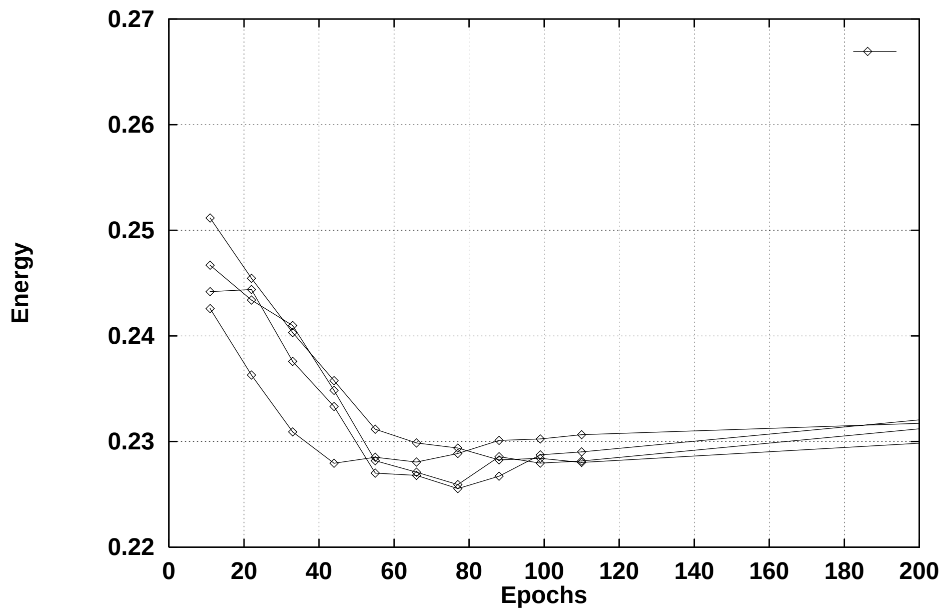
If the "gradient-multiplier" G_{decr} in Figure 9 is 0.5, the λ_{min} that minimizes the parabola is less than λ.

The algorithm is very effective for the benchmark neural net learning task: while the total number of iterations (80) is larger than that of CG with exact searches, but the number of function and gradient calls per iteration for OSS is about 1.1, so that the final CPU-time is reduced by a factor of about 8 with respect to the "library" conjugate-gradient. To check the robustness of this technique with respect to the initial learning rate, it has been changed from 10^{-5} to 10^{-4}. In this case the best generalization is obtained at 50 iterations, but the average total number of functions and gradients evaluated are 72 and 63, respectively: the number of quadratic interpolations executed increases but the final amount of computation is comparable.

The generalization results are shown in Figure 11.

```
function fast_line_search (d⃗)
comment
Search for a new weight vector starting from the current configuration w⃗_curr and moving along
direction d⃗. d_1 is the directional derivative of E along d⃗. G_decr is a constant equal to 0.5,
used to multiply the directional derivative. The constant MAX_TRIALS is equal to 10.
L_incr is a constant equal to 1.1. L_decr is a constant equal to 0.5.
begin
d_1 = g⃗ · d⃗
If d⃗ is not a descent direction (d_1 > 0), reset it to the negative gradient g⃗:
if d_1 > 0 then
    begin
    d⃗ := − g⃗;  d_1 := g⃗ · d⃗
    end
Save the value of E and the gradient corresponding to w⃗_curr:
E_saved := E;
ε := L_incr · ε
ok := FALSE;  trials := 0
repeat
    begin
    trials := trials + 1
    w⃗ := w⃗_curr + ε · d⃗
    E := E(w⃗)
    if E < (E_saved + G_decr · d_1 · ε) then
        ok := TRUE
    else
        begin
        ε_quad := parabola_minimizer (E_saved, d_1, f)
        w⃗ := w⃗_curr + ε_quad · d⃗
        E := E(w⃗)
        if E < (E_saved + G_decr · d_1 · ε_quad) then
            begin
            ok := TRUE
            ε := ε_quad
            end
        else
            ε := L_decr · ε
        end
    end
until ok = TRUE or trials > MAX_TRIALS
if ok = TRUE then
    begin
    p⃗ := ε · d⃗
    w⃗_curr := w⃗
    g⃗ := ∇_w⃗ E(w⃗)  (Only the backward pass of backpropagation is needed at this point.)
    ε̄ := 0.9 · ε̄ + 0.1 · ε
    end
fast_line_search := ok;
end
```

Figure 10: Pseudo-code description of the One-Step Secant algorithm. Part II



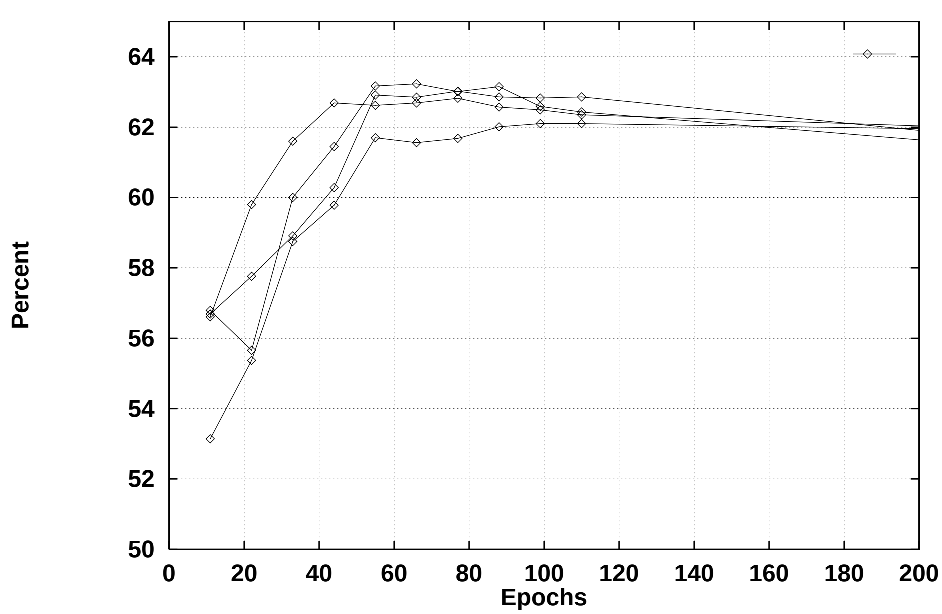


Figure 11: Generalization results for the One-Step Secant algorithm. Average squared error (above) and recognition accuracy (below)

4 Methods with No Derivatives

In the general context of optimization there are situations where partial derivatives cannot be used because, for example, the function is not differentiable or because computing the derivatives is too expensive. This fact motivates the study of optimization techniques based only on the knowledge of function values. In this section we investigate variants of the adaptive random search algorithm presented in [12],[9] and based on the theoretical framework of [28].

The general scheme starts by choosing an initial point in the configuration space and an initial ***search region*** surrounding it, and it proceeds by iterating the following steps:

- (i) a new candidate point is generated by sampling the search region according to a given probability measure.
- (ii) the search region is adapted according to the value of the function at the new point. It is compressed if the new function value is greater than the current one (unsuccessful sample) or expanded otherwise (successful sample).
- (iii) if the sample is successful, the new point becomes the current point, and the search region is moved so that the current point is at its center for the next iteration.

For effective implementations it is sufficient to adopt simple search regions around the current point, for example regions defined by ***boxes*** (i.e., regions with edges given by a set of linearly independent vectors) with uniform probability distributions inside the box. In this case generating a random displacement is simple: the basis vectors are multiplied by random numbers in the real range (-1.0, 1.0) and added: $\vec{\delta} = \sum_j rand \times \vec{b}_j$. Because of the brisk movements of the search trajectory our versions of stochastic search are called "Shaker".

In addition to the above framework, it is possible to accelerate the convergence speed by storing in memory the most recent steps executed in the configuration space and by using them to determine the current movement, see [10].

A first improvement of the algorithm is obtained by using the 'double-shot strategy': if the first sample $\vec{w} + \vec{\delta}$ is not successful, the specular point $\vec{w} - \vec{\delta}$ is considered. This choice drastically reduces the probability of generating two consecutive unsuccessful samples [9]. The two variants presented contain the "double-shot" strategy and differ by the way in which the boxes are adapted during the search. The use of boxes with sides parallel to the coordinate axes permits an efficient and simple updating but the algorithm may encounter inefficiencies if the "valleys" in the search space are not parallel to the axes. In this case the motion of current point is forced to zig-zag around the "trend" direction (given by the "bottom of the valley") because the direction has to be produced by elementary displacements along the coordinate axes. To overcome this pathological behavior two modifications are possible. In the first modification (see Section 4.1) the box edges remain parallel to the coordinate axes and a mechanism is added in order to follow the "trend" direction by adaptively averaging the previous displacements. In the second the requirement that the box edges are parallel to the coordinate axes is relaxed so that the frames can be compressed or expanded along arbitrary directions by using affine transformations (see Section 4.2),

4.1 Inertial Shaker

This variant is presented in [10], here we summarize the main features. Figure 12 describes the basic procedure; it is composed of a block in which the double-shot strategy is applied iteratively to each component and the region defined by the vector $\vec{\eta}$ is adapted (see the procedure ***double_shot_on_all_components*** in Figure 13).

If a new successful sample is found, the displacement $\vec{\delta}$ is returned and added to a circular buffer. The most recent displacements stored in the buffer are used to build a new tentative displacement by using the averaging process illustrated in Figure 12. Let us note that recent displacements are weighted more than old ones. If the new trial is successful the "temporal window" used for averaging is enlarged by increasing the time constant σ. Similarly, the factor α that regulates the step size is increased (see Figure 12). If the trial is unsuccessful, σ and α are reduced.

This trend identification by averaging previous displacements and therefore reducing the high frequency oscillations is analogous to the use of the "momentum" term in back propagation [27]. The Inertial Shaker Algorithm has been tested for Radial Basis Function networks in [12] and for

procedure *inertial_shaker*
comment
Starting from an initial search region around the starting point defined by the N-dimensional parallelepiped whose j-th side has length b_j, compute iteratively new points by applying the 'double-shot strategy' followed by the averaging of the previous displacements (trend identification).
The constants L_{incr} and L_{decr}, used to enforce or reduce the trend identification, are equal to 1.1 and 0.9 respectively.
begin
Choose randomly $\vec{w}_{curr}$. Set time:=0 and increment it on every evaluation of *E*.
Δ and τ are two circular buffers of size circular_buffer_length.
Initialize the variables α and σ to 0.5 and 15.0 respectively.
Initialize $\vec{b}$: $b_k := 0.1 \cdot$ weight_range.
while convergence_criterion_is_not_satisfied **do**
    begin
    double_shot_on_all_components ($\vec{\delta}$, success_flag)
    if success_flag = TRUE **then**
        begin
        $\vec{w}_{curr} := \vec{w}_{curr} + \vec{\delta}$
        Insert $\vec{\delta}$ and time into the two circular buffers Δ and τ
        find_trend ($\vec{\delta}$)
        if $E(\vec{w}_{curr} + \vec{\delta}) < E(\vec{w}_{curr})$ **then**
            begin
            $\vec{w}_{curr} := \vec{w}_{curr} + \vec{\delta}$
            $\alpha := \alpha \cdot L_{incr}$; $\sigma := \sigma \cdot L_{incr}$
            end
        else
            $\alpha := \alpha \cdot L_{decr}$; $\sigma := \sigma \cdot L_{decr}$
        end
    end
end

procedure *find_trend* ($\vec{\delta}$)
comment
Compute the trend $\vec{\delta}$ by averaging the previous displacements (stored in Δ) multiplied by weights that decrease gaussianly with respect to the elapsed time (stored in τ).
The variables σ and α determine the number of used displacements and the amplification factor, respectively, and are dynamically adjusted during the search (see the procedure *inertial_shaker*).
begin

$$\vec{\delta} := \alpha \cdot \frac{\sum_{i=m_1}^{m_2} \vec{\Delta}_i e^{-\frac{(time-\tau_i)^2}{\sigma^2}}}{\sum_{i=m_1}^{m_2} e^{-\frac{(time-\tau_i)^2}{\sigma^2}}}$$

where:
- m_1 is such that $\vec{\Delta}_{m_1}$ is the last displacement evaluated.
- m_2 is such that the exponential term in the sum is not negligible (i.e., greater than 10^{-4}). and such that $(m_2 - m_1) <$ circular_buffer_length.

end

Figure 12: Pseudo-code description of the Inertial Shaker algorithm. Part I

```
procedure double_shot_on_all_components (δ⃗; success_flag)
comment:
On every component of w⃗_curr apply the 'double-shot strategy' and adjust the vector b⃗
by compressing or expanding its components depending on the trial success.
On return δ⃗ contains the displacement and success_flag is TRUE if a better point
is found, FALSE otherwise. rand returns a random number in the range (-1.0, 1.0).
The two real constants ρ_comp and ρ_exp are equal to 0.5 and 2.0.
begin
success_flag := FALSE
w⃗ := w⃗_curr
for i:=1 to N do
    begin
    E := E(w⃗)
    r := rand ·b_i
    w_i := w_i + r
    if E(w⃗) > E then
        begin
        w_i := w_i − 2 · r
        if E(w⃗) > E then
            begin
            b_i := ρ_comp b_i
            w_i := w_i + r
            end
        else
            begin
            b_i := ρ_exp b_i
            success_flag := TRUE;
            end
        end
    else
        begin
        b_i := ρ_exp b_i
        success_flag := TRUE;
        end
    end
if success_flag = TRUE then
    δ⃗ := w⃗ − w⃗_curr
end
```

Figure 13: Pseudo-code description of the Inertial Shaker algorithm. Part II

multilayer perceptrons (parity and dichotomy tasks) in [7].

The generalization results obtained are shown in Figure 12.

4.2 Affine Shaker

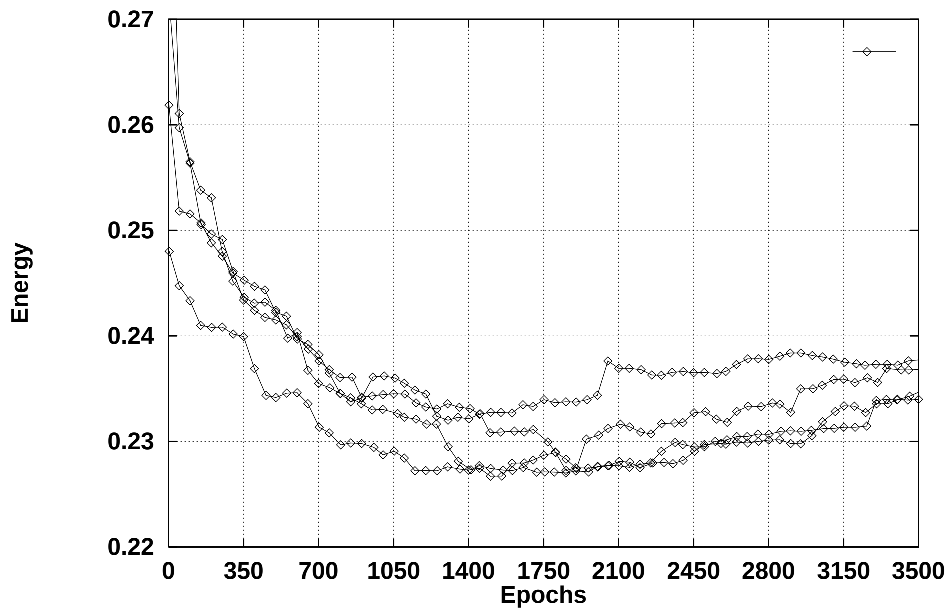
In the previous Inertial Shaker algorithm the averaging step was devised to cope with box axes parallel to the coordinate axes. In the novel Affine Shaker version the cause of the problem is eradicated by allowing arbitrary frames. The term "affine" derives from the use of affine transformations to modify the local search region, that is translated when a successful sample is found and expanded or compressed along the trial direction. These transformations permit adapting the frame so that the search region can be elongated along arbitrary success directions, and compressed along unsuccessful ones. In this way the generation of samples is tuned to properties of the E surface.

The main procedure of the Affine Shaker is illustrated in Figure 15. The algorithm is composed of a block where a new candidate is found by the 'double-shot strategy' starting from the set of linearly independent vectors Γ (see the procedure ***double_shot*** in Figure 15). The box is stretched along the direction of the last displacement if a successful candidate is found, if not it is compressed. In detail, if $\vec{\delta}$ identifies the direction, the box edges are transformed according to the following linear operator:

$$P \equiv I + (\rho - 1) \frac{\vec{\delta}\,\vec{\delta}^T}{\|\vec{\delta}\|^2} \tag{23}$$

This choice is motivated by the fact that P dilates every vector parallel to $\vec{\delta}$ and leaves unchanged every vector perpendicular to $\vec{\delta}$. The volume of the box is changed by the factor ρ.

If the generalization results shown in Figure 17 are compared with the curves obtained with the Inertial Shaker one notes that the new algorithm reaches the maximum generalization with less iterations. In addition, after analysing the evolution up to 3500 iterations, the over-training effect is weaker for the new algorithm: on average the accuracy degrades by 1.6% with the Inertial Shaker and by 0.7% with the Affine Shaker.



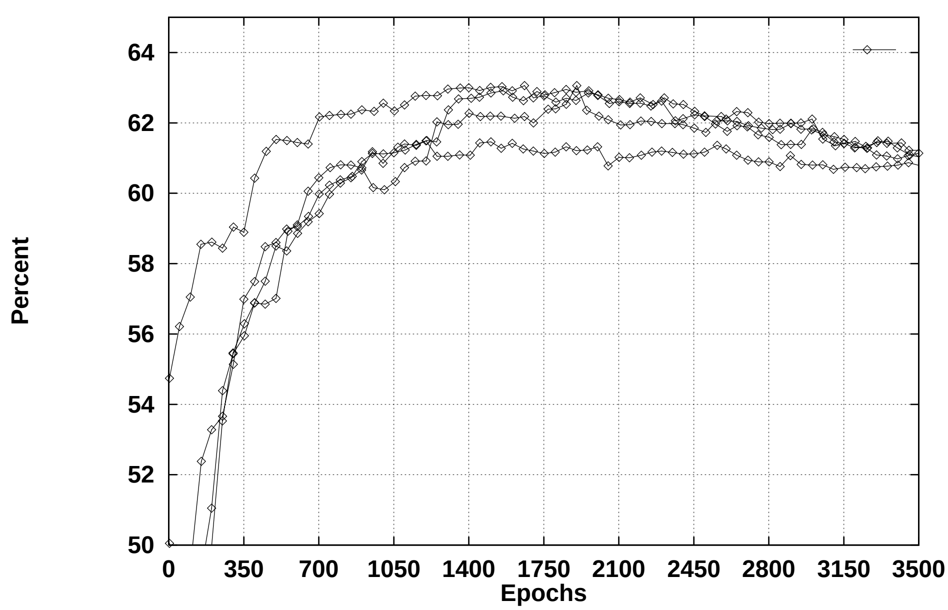


Figure 14: Generalization results for the Inertial Shaker algorithm. Average squared error (above) and recognition accuracy (below)

procedure *affine_shaker*
comment:
Search for the optimum, using the 'double shot strategy' and applying affine trasformations (expansion or compression) on the search region sorrounding the point.
ρ_{exp} and ρ_{comp} are two constants equal to 2.0 and 0.5, respectively.
begin
Choose randomly the starting point $\vec{w}_{\mathrm{curr}}$.
Choose the set of n linearly independent vectors $\Gamma := \{\vec{b}_k\}$ defining the initial search region $\Omega(\vec{w}_{\mathrm{curr}}) = \{\vec{w} : \vec{w} = \vec{w}_{\mathrm{curr}} + \sum_{j:=1}^{N} \mu_j \vec{b}_j \text{ where } |\mu_j| < 1 \}$.
Our choice is: $\vec{b}_k := l_k \cdot \vec{e}_k$, where $\{\vec{e}_k\}$ is the canonical basis of R^N
and l_k are constants equal to weight_range $\cdot 0.1$.
while convergence_criterion_is_not_satisfied **do**
    begin
        double_shot($\vec{w}_{\mathrm{curr}}$, Γ, $\vec{\delta}$, success_flag)
        if succes_flag = TRUE **then**
            begin
            $\vec{w}_{\mathrm{curr}} := \vec{w}_{\mathrm{curr}} + \vec{\delta}$
            modify_box($\vec{\delta}$, Γ, ρ_{exp});
            end
        else
            modify_box($\vec{\delta}$, Γ, ρ_{comp});
    end
end

procedure *double_shot* ($\vec{w}_{\mathrm{curr}}$; Γ; $\vec{\delta}$; success_flag)
comment:
A sample point in $\Omega(\vec{w}_{\mathrm{curr}})$ is computed using the 'double-shot strategy'.
If the sample is a better than $\vec{w}_{\mathrm{curr}}$ then, on return, $\vec{\delta}$ is set to the displacement and success_flag to TRUE, else success_flag is set to FALSE. *rand* returns a random number in (-1.0,1.0).
Γ is the set of linearly independent vectors $\{\vec{b}_k\}$ defining Ω (see the procedure *affine_shaker*).
begin
$\vec{\delta} := \sum_{j:=0}^{N} rand \cdot \vec{b}_j$
$\vec{w} := \vec{w}_{\mathrm{curr}} + \vec{\delta}$
if $E(\vec{w}) > E(\vec{w}_{\mathrm{curr}})$ **then**
    begin
    $\vec{\delta} := -\vec{\delta}$
    $\vec{w} := \vec{w}_{\mathrm{curr}} + \vec{\delta}$
    if $E(\vec{w}) > E(\vec{w}_{\mathrm{curr}})$ **then** success_flag := FALSE **else** success_flag := TRUE
    end
else
    success_flag := TRUE
end

Figure 15: Pseudo-code description of the Affine Shaker algorithm. Part I

procedure *modify_box* $(\vec{\delta}; \Gamma; \rho)$
comment:
Compress (or expand) the region defined by $\Gamma = \{\vec{b}_k\}$ by a factor ρ (expansion if $\rho > 1.0$, compression if $0.0 < \rho < 1.0$) along the direction of $\vec{\delta}$.
I is the identity matrix.
begin
Compute the compression (expansion) operator:

$$P := I + (\rho - 1)\,\frac{\vec{\delta} \cdot \vec{\delta}^{T}}{\|\vec{\delta}\|^2}$$

Compute the new basis:
for k:=1 **to** N **do**
$\vec{b}_k := P \cdot \vec{b}_k$
end

Figure 16: Pseudo-code description of the Affine Shaker algorithm. Part II

5 Discussion

The results of all algorithms tested on the bottom discrimination task have been collected in Figure 18 (averages of 10 trials and standard deviation). The percent accuracy with the corresponding average squared error is shown for a number of iterations corresponding to the maximum average generalization (apart from the finite test interval).

To compare the different results let us recall that the Affine and Inertial Shaker require only the forward pass, so that the CPU time required for each iteration is about 50% of the time required by one step of batch backpropagation (the exact CPU time naturally depends on the specific software and processor). In addition, each iteration of on-line backpropagation requires about a factor of 10^{-4} less CPU time than the batch version. Finally, each iteration of CG and OSS requires different numbers of function and gradient evaluations (written in parentheses below the number of iterations).

It is apparent that on-line BP is the fastest technique, provided that the learning rate is suitable and that oscillations in the results do not cause a large standard deviation (this effect is evident for ϵ larger than 0.1). The OSS secant method requires about double CPU time, a price that one can

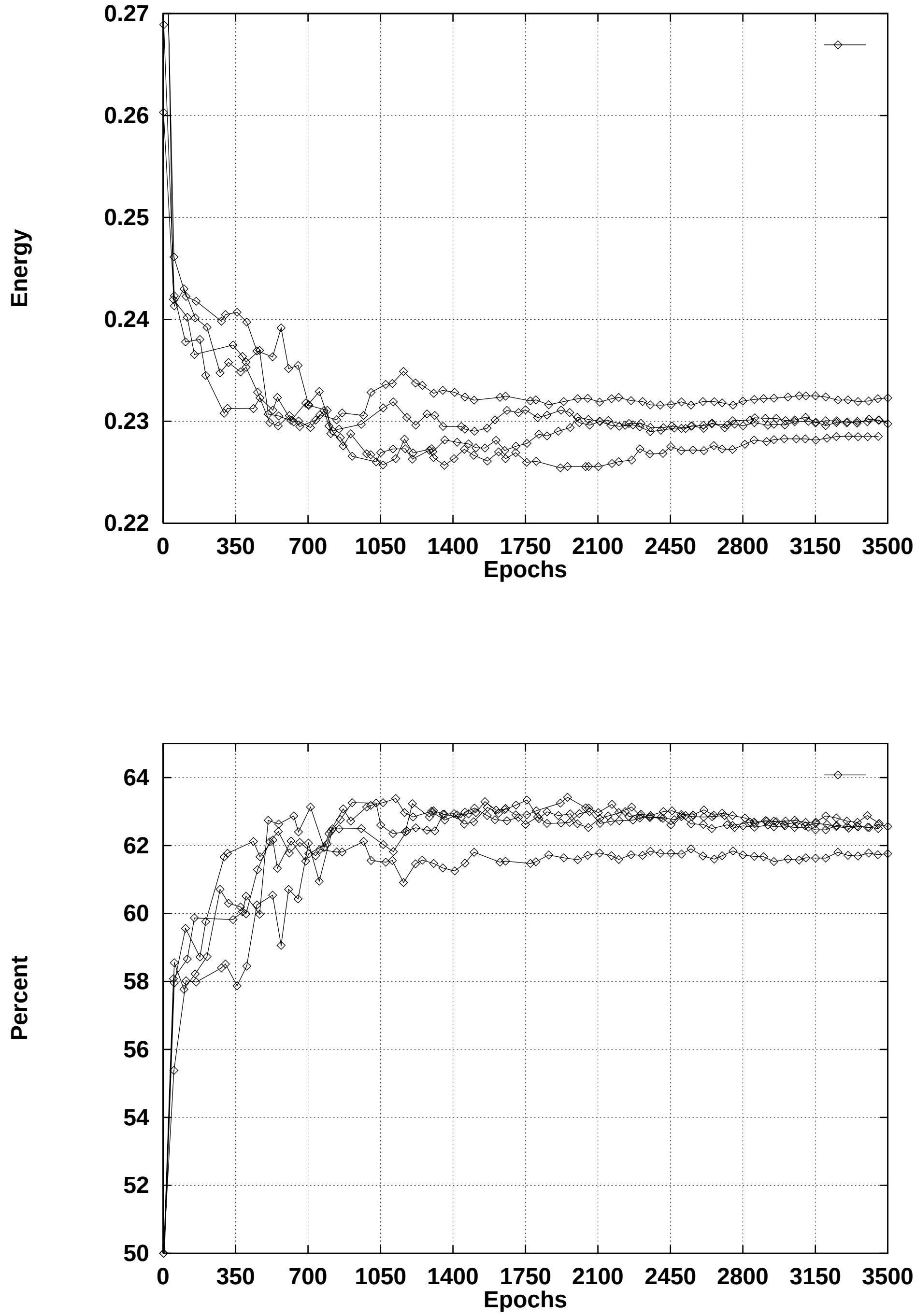


Figure 17: Generalization results for the Affine Shaker algorithm. Average squared error (above) and recognition accuracy (below)

Experimental Results					
Method	**Iterations**	**Percent**		**Energy**	
		(average)	(std.dev.)	(average)	(std.dev.)
Affine	1000	61.5	0.5	0.226	0.001
Inertial	1500	61.5	0.5	0.228	0.002
One Step Secant	80 [87-87]	62.5	0.5	0.227	0.002
Conjugate Gradient (Fletcher-Reeves)	20 [645-503]	62.5	1.0	0.227	0.002
Conjugate Gradient (Polak-Ribiere)	20 [643-495]	62.5	0.5	0.228	0.002
Bold Driver	500	62.0	1.0	0.229	0.003
On Line ($\varepsilon = 10^{-3}$)	400000	62.0	0.5	0.229	0.001
On Line ($\varepsilon = 10^{-1}$)	400000	61.5	1.0	0.233	0.003

Figure 18: Comparison of test results on the "bottom discrimination" case.

afford to avoid a trial-and-error selection of the learning rate and to ensure a smoother evolution of learning. Note that the number of average function and gradient evaluations per iteration is 1.08; therefore a very small number of quadratic interpolations is actually executed. Using one-step BFGS to define the search direction is the crucial element, while the quadratic interpolation acts as a "safety" addition.

The "library" conjugate gradient techniques require less iterations but the number of function and gradient calls is actually larger than those required by the simple BD technique.

The performance of the Affine Shaker is comparable to that of BD, a surprising result for a technique based only on function evaluations. The method can be the right choice for dedicated VLSI implementations, or, in general, when derivative computation is impossible (non-differentiable E) or expensive. The fact that effective learning in multi-layer perceptrons can be executed without derivatives agrees with the results described in [6], where the *reactive TABU* algorithm for combinatorial optimization is used.

A promising avenue for research is that of improving the selection of the search direction with a *compromise* between the on-line technique, in which where the direction is derived from a single pattern, and the batch technique, in which the entire set of training patterns is considered, see for example the "blockupdate" suggested in [22]. This compromise can be tested also in the stochastic techniques like the Affine Shaker.

Acknowledgment

We thank Prof. Roberto Odorico for making available the events produced by the COJETS generator as a benchmark training and test set, INFN for allowing the use of its hardware facilities and S.Struthers for her kind assistance.

References

[1] A.-I. Amari, Neural theory of association and concept-formation, Biological Cybernetics 26 (1977) 175–185.

[2] E. Barnard and R. A. Cole, A neural-net training program based on conjugate-gradient optimization, Tech. Rep. CSE 89-014, Oregon Grad. Inst. of Sci. and Tech, (July 1989).

[3] R. Battiti, First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method, Neural Computation 4 (2) (1992) 141–166.

[4] R. Battiti, Accelerated Back-Propagation Learning: Two Optimization Methods, Complex Systems 3 (4) (1989) 331–342.

[5] R. Battiti and F. Masulli, BFGS Optimization for Faster and Automated Supervised Learning, Proceedings of the International Neural Network Conference (INNC 90) - Paris - France, 757–760.

[6] R. Battiti and G. Tecchiolli, Training Neural Nets with the Reactive Tabu Search, Preprint Dip. Mat. Universita' di Trento, (1993).

[7] R. Brunelli, On training neural nets through stochastic minimization, Tech. Rep. 9212-06, IRST, Trento, Italy (1992).

[8] C. Bishop, Exact Calculation of the Hessian Matrix for the Multilayer Perceptron, Neural Computation 4 (1992) 949–501.

[9] R. Brunelli and G. Tecchiolli, On random minimization of functions, Biol. Cybern. 65 (1991) 501-506.

[10] R. Brunelli and G. Tecchiolli, Stochastic minimization with adaptive memory, Journal of Computational and Applied Mathematics, in press.

[11] E. R. Caianiello, Outline of a theory of thought processes and thinking machines, J. Theor. Biol. 1 (1961) 204–235.

[12] B. Caprile, F. Girosi and T. Poggio, A nondeterministic method for multivariate functions minimization, in: Parallel Architectures and Neural Networks, E.R. Caianiello ed., (World Scientific, Singapore, 1991).

[13] C. de Groot and D.Wurtz, Analysis of univariate time series with connectionist nets. A case study of two classical examples, Neurocomputing 3 (4) (1991) 177–192.

[14] J.E. Dennis and R.B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations (Prentice Hall, 1983).

[15] P.E. Gill, W. Murray and M. H. Wright, Practical Optimization (Academic Press, 1981).

[16] G. H. Golub and C. F. Van Loan, Matrix Computations, second edition (The John Hopkins University Press, Baltimore, 1989).

[17] T.D. Gottshalk and R. Nolty, Identification of Physics Processes Using Neural Network Classifiers, Report CALT-68-1680, California Institute of Technology, Pasadena, CA (1989).

[18] E. M. Johansson, F. U. Dowla and D. M. Goodman, Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method, Preprint UCRL-JC-104850, Lawrence Livermore Nat. Lab. (Sept. 1990).

[19] A. Lapedes and R. Farber, A self-optimizing, nonsymmetrical neural net for content addressable memory and pattern recognition, Physica 22 D (1986) 247–259.

[20] P. Mazzanti and R. Odorico, Bottom Jets Recognition by Neural Networks and Statistical Discriminants: a Survey, Report DFUB 92/15 (1992), University of Bologna, Dept. Physics, Zeitschrift für Physik C , in press.

[21] M. F. Møller, A scaled conjugate gradient algorithm for fast supervised learning, Comp. Sci. Dept., University of Aarhus, preprint, (Nov. 1990).

[22] M. F. Møller, Supervised Learning on Large Redundant Training Sets In: Neural Networks for Signal Processing II, eds. S.Y. Kung, F. Fallside, J.A. Sorenson and C.A. Kamm, Proceedings of the 1992 IEEE-SP Workshop (1992) 79–89.

[23] R. Odorico, COJETS: a Monte Carlo Program Simulating QCD in Hadronic Production of Jets and Heavy Flavors with Inclusion of Initial QCD Bremsstrahlung, Comput. Phys. Commun. 32 (1984) 139–172.

[24] R. Odorico, COJETS 6.23: a Monte Carlo Simulation Program for Antiproton-Proton, Proton-Proton Collisions and Electron-Positron Annihilation, Comput. Phys. Commun. 72 (1992) 238–248.

[25] T. Poggio and L. Stringa A Project for an Intelligent System: Vision and Learning, International Journal of Quantum Chemistry, 42, (1992) 727–739.

[26] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Wetterling, Numerical Recipes in C (Cambridge University Press, 1988).

[27] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart, and J. L. McClelland, eds., Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations (MIT Press, Cambridge, 1986).

[28] F.J. Solis and R. J-B. Wets, Minimization by random search techniques, Mathematics of Operations Research 6(1) (1981) 19–30.

[29] S. Saarinen, R. Bramley and G. Cybenko, The numerical solution of neural network training problems, SIAM Journal of Statistical and Scientific Computing (May, 1993), in press.

[30] R. Serra and G. Zanarini, Complex Systems and Cognitive Processes (Springer-Verlag, Berlin, 1990).

[31] D. F. Shanno, Conjugate gradient methods with inexact searches, Mathematics of Operations Research 3-3 (1978) 244–256.

[32] T.P. Vogl, J.K. Mangis, A.K. Rigler, W. T. Zink and D. L. Alkon, Accelerating the Convergence of the Back-Propagation Method, Biological Cybernetics 59 (1988) 257–263.

[33] Watrous, R. Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization. Technical Report MS-CIS-87-51, Univ. of Penn 1987.