Waypoint averaging and step size control in learning by gradient descent

***G. Papari*[1], *K. Bunte*[2], *M. Biehl*[2,3]**

Acknowledgements: This work was supported by the *Nederlandse Organisatie voor Weetenschappelijke Onderzoek* (NWO) under project code 612.066.620.

Abstract

We introduce a modification of batch gradient descent, which aims at better convergence properties and more robust minimization. In the course of the descent, the procedure compares the performance of the actual configuration with that of a gliding average over the most recent positions. If the latter corresponds to a lower value of the optimization objective, minimization proceeds from there and the step size of the descent is decreased.
Here we present the prescription from a practitioner's point of view and refrain from a detailed mathematical analysis. First, the method is illustrated in terms of a low dimensional example. Moreover, we discuss its application in the context of machine learning, examples corresponding to multilayered neural networks and a recent extension of Learning Vector Quantization (LVQ) termed Matrix Relevance LVQ.

1 Introduction

Gradient based minimization is one of the most popular, basic techniques in non-linear optimization [18]. While many, more sophisticated methods are also gradient based, plain gradient descent faces a number of significant problems. First of all, the success of steepest descent depends crucially on the choice of an appropriate magnitude of the

[1]National Institute of Research in Informatics and Automatics (INRIA)
Department CLIME, BP 105, 78153 Le Chesnay Cedex, France
[2]Johann Bernoulli Institute for Mathematics and Computer Science
University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands
[3]E-mail: m.biehl@rug.nl

update step. Too careful updates will cause slow convergence, while large steps may result in oscillatory or even divergent behavior.

Methods for automated step size control and so-called *line-search* procedures have been designed which can overcome this difficulty to a large extent. Similarly, in the well-known *conjugate gradient descent* a coefficient is determined which controls the superposition of two orthogonal descent steps [18]. Higher order methods which employ second or further derivatives include, among others, Newton and Quasi-Newton methods. Arguably, the latter play the most important role in practical optimization of non-convex non-linear cost functions, nowadays. Frequently, these methods are computationally expensive and difficult to implement in high-dimensional search spaces. In addition, they often require the tuning of algorithm parameters which further complicate their use in practice.

In particular in the specific context of machine learning, plain gradient descent has played a key role and continues to do so for several reasons. Gradient descent gained significant importance when multi-layered neural networks were introduced and studied, initially. The availability of simple and efficient implementations of gradient descent, e.g. the well-known *backpropagation of error* [19, 12, 17, 13, 5], contributed immensely to the popularity of neural networks and machine learning in general.

To date, gradient descent is a popular tool in many machine learning tasks that can be formulated in terms of, frequently non-convex, non-linear optimization problems. Due to its simplicity and flexibility, gradient descent is often the first choice in initial investigations of novel learning paradigms. It has been employed in, both, supervised and unsupervised learning. Examples for the former comprise the already mentioned training of multi-layered neural networks by means of backpropagation and, more recently, prototype-based Learning Vector Quantization and variants [11]. Competitive learning in Vector Quantization [17] and cost function based variants of Neural Gas [15, 3], constitute important examples for the application of gradient descent in unsupervised learning.

In the machine learning domain, most frequently, the cost function and, thus, its gradient can be written as a sum over the available example data. This facilitates the use of a particularly simple and efficient scheme termed *stochastic gradient descent*, which is also known as the *Robbins Monro* procedure [20, 13] in a more general context. Here, the actual gradient is approximated by the contribution of a single training example. The noise introduced by its random selection is believed to be beneficial, for instance with respect to escaping local minima. For a discussion of various training prescriptions which are based on the stochastic approximation of gradients, see [7].

On the other hand, *batch* gradient procedures make use of all examples in every iteration, which increases the computational effort per step, but may be advantageous in terms of efficiency.

Generic problems of gradient descent are also present, and sometimes particularly pronounced, in both variants of gradient descent training. While the effect of local minima on the actual performance of the resulting system is not always clear, their presence certainly complicates the training process. Local minima result in, for instance, high

sensitivity to initial conditions of the training process.

Flat regions in the search space, where the gradient of the cost function displays low magnitude can also constitute a problem in practice. They can result in so-called quasi-stationary *plateau states* which can drastically slow down the learning process and, frequently, dominate the shape of learning curves in gradient based training. For a mathematical analysis of this phenomenon, borrowing concepts from statistical physics, see for instance [10, 14, 6, 9, 4].

As a consequence, many modifications of plain gradient descent have been introduced and investigated within the machine learning community. The choice of appropriate learning rates and learning rate schedules plays a key role, obviously. For stochastic gradient descent, for instance, exact criteria are known for schedules which realize convergence to a (local) minimum. In practice, one has to compromise between the desired approach to a potentially global minimum on the one hand and constraints on the tolerated computational effort on the other.

The problem of flat regions of the cost function, in which steepest descent without normalization of the gradient is slow, has attracted considerable interest in the machine learning community. One the most popular extensions of gradient based training introduces a memory term which is supposed to facilitate persistent moves along previously found directions of descent. The term *momentum* has been coined for this popular concept [19, 12, 17]. Other modifications of gradient descent concern the design of so-called *well-behaved cost functions* which modify the original objective, aiming at fast initial training and better convergence properties, see for instance [17].

The use of higher order methods has been explored also in the context of machine learning. For reviews, concrete examples, and further references we suggest to consult, for instance, [7, 17, 12]. Obviously, the evaluation or estimation of higher order derivatives poses a practical problem in high-dimensional spaces and limits the usefulness of the approach in many learning problems.

In the context of stochastic gradient descent, an averaging procedure has been suggested which does not modify the descent itself, but interprets the mean over all performed descent steps as the actual outcome of training [16, 1]. Obviously, this will reduce the influence of random fluctuations while keeping the presumed advantages of stochastic descent. Indeed, the approach has been shown to yield favorable convergence properties in [1].

In the following we suggest an approach which combines the basic idea of *waypoint averages*, here over a limited history, with an appropriate step size adaption. It provides a conceptually simple and computationally efficient extension of steepest descent. It is easy to implement and bears the promise to yield robust performance in, for instance, practical learning problems or more general optimization tasks.

In this report we focus on a heuristic motivation and present the algorithm from a practitioner's point of view, with particular emphasis on machine learning applications. More mathematical aspects of the method will be presented elsewhere. We illustrate the approach in terms of low-dimensional optimization problems as well as an example machine learning problem.

2 The Algorithm

First we consider the case of an objective function E which depends on a d-dim. vector $\mathbf{x} \in I\!R^d$. A gradient descent procedure, initialized in $\mathbf{x}_o$, generates a sequence of positions $\mathbf{x}_t$ by an iteration of the form

$$\mathbf{x}_{t+1} \;=\; \mathbf{x}_t \;-\; \alpha_t \, \frac{\nabla E_t}{|\nabla E_t|}. \tag{1}$$

Here and in the following we use the shorthands $\nabla E_t \;=\; \nabla E|_{\mathbf{x}=\mathbf{x_t}}$. Note that the gradient is normalized in Eq. (1). Hence, α_t controls explicitly the step length in terms of Euclidean distance in $I\!R^d$: $|\mathbf{x}_{t+1} - \mathbf{x}_t| = \alpha_t$. Accordingly, we will refer to α_t as the *step size* at iteration step t. The related quantity $\eta_t \;=\; \alpha_t / \left|\nabla E_t\right|$ corresponds to the *learning rate* in standard machine learning jargon, i.e. a pre-factor of the unnormalized gradient.

In order to ensure convergence one has to set the learning rate or step size, respectively, small enough. For constant $\eta_t = \eta$ it is straightforward to work conditions for the convergence of gradient descent close to a (local) minimum $\mathbf{x}^*$. Let us assume that we can expand E as

$$E(\mathbf{x}) \;\approx\; E(\mathbf{x}^*) \;+ \frac{1}{2} \; (\mathbf{x} - \mathbf{x}^*)^\top \; H^* \; (\mathbf{x} - \mathbf{x}^*) \tag{2}$$

where the elements of the Hesse-Matrix H^* are given by $H^*_{ij} \;=\; \left. \frac{\partial^2 E}{\partial x_i \, \partial x_j} \right|_{\mathbf{x}=\mathbf{x}^*}$.

The largest eigenvalue λ_{max} of H^* corresponds to the largest curvature observed in $\mathbf{x}^*$. One can show that for $\eta < 2/\lambda_{max}$ the deviation $|\mathbf{x}_t - \mathbf{x}_*|$ vanishes as $t \to \infty$. However, in practical situations, the properties of the unknown minimum are not known and H^* itself is not available. A variety of schemes exist, which resort to the evaluation of the local Hesse matrix H for automatic step size adaptation in machine learning, see [7] for further references. More frequently, simple heuristic *annealing schemes* are used which reduce η_t explicitly with time, see [20, 13, 17, 7] for a discussion and examples. Note that these schemes inevitably introduce a number of algorithm parameters which have to be fine-tuned to the concrete practical learning problem at hand.

Here we present a simple and robust extension of gradient descent which improves convergence by considering *waypoint averages* over the latest iteration steps and implements an efficient step size adaptation at the same time. It does not require the costly evaluation of higher order derivatives and the number of additional control parameters is very small compared to some of the other approaches mentioned above.

Waypoint averaging and step size adaptation
The iteration is initialized in $\mathbf{x}_o$ and the initial step size is α_o. First, a number k of

unmodified gradient steps is performed, i.e.

$$\mathbf{x}_{j+1} \;=\; \mathbf{x}_j \;-\; \alpha_j \,\frac{\nabla E_j}{|\nabla E_j|} \quad \text{for } \; j = 0, 1, 2, \ldots, k-1 \quad \text{with} \quad \alpha_j = \alpha_o. \tag{3}$$

Thereafter, the iteration proceeds as described in the following:

1. evaluate the *tentative* gradient step

$$\widetilde{\mathbf{x}}_{t+1} \;=\; \mathbf{x}_t \;-\; \alpha_t \,\frac{\nabla E_j}{|\nabla E_j|} \;\text{ and }\; E(\widetilde{\mathbf{x}}_{t+1}) \tag{4}$$

2. calculate the *waypoint average* over the previous k steps:

$$\widehat{\mathbf{x}}_{t+1} \;=\; \frac{1}{k}\sum_{i=0}^{k-1} \mathbf{x_{t-i}} \;\text{ and }\; E(\widehat{\mathbf{x}}_t) \tag{5}$$

3. determine new position and new step size as

$$\begin{cases} \mathbf{x}_{t+1} = \widetilde{\mathbf{x}}_{t+1} \;\text{ and }\; \alpha_{t+1} = \alpha_t & \text{if } \; E(\widetilde{\mathbf{x}}_{t+1}) \le E(\widehat{x}_{t+1}) \\ \mathbf{x}_{t+1} = \widehat{\mathbf{x}}_{t+1} \;\text{ and }\; \alpha_{t+1} = r \cdot \alpha_t & \text{else.} \end{cases} \tag{6}$$

with the parameter $r < 1$.

As long as the plain gradient descent step yields a position which corresponds to lower costs than the *waypoint average* $\widehat{\mathbf{x}}_{t+1}$ over the last k steps, the iteration proceeds unmodified.

On the contrary, $E(\widehat{\mathbf{x}}_{t+1}) < E(\widetilde{\mathbf{x}}_{t+1})$ signals that the procedure has *overshot* and displayed oscillatory behavior because the step size has been *too large* for smooth convergence. As a consequence, one may expect that the positions $\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-k+1}$ fluctuate about a local minimum and the *waypoint average* should provide a better estimate than the tentative $\widetilde{\mathbf{x}}_{t+1}$. In this case, the iteration proceeds from $\widehat{\mathbf{x}}_{t+1}$ and the step size is reduced by a factor $r < 1$.

In a forthcoming publication we will discuss favorable settings of the parameter r. In addition, several extensions and modifications of the basic prescription are possible. For instance, an additional parameter $q > 1$ could be introduced to increase the step size as $\alpha_{t+1} = q \cdot \alpha_t$ whenever the *tentative* step is accepted, thus avoiding slow convergence due to inappropriately small step sizes. Here we restrict the discussion to the case $q = 1$ and refer to forthcoming studies for the discussion of the extension.

Figure 5 shows a simple example in $d = 2$ dimensions and illustrates the method by comparing updates with constant step size and the procedure with waypoint averaging and step size control.

3 A machine learning example

Frequently, subsets of variables can be identified which play qualitatively different roles in the optimization problem with significantly different gradient magnitudes and curvatures of E. In the suggested descent procedure, meaningful groups of variables can

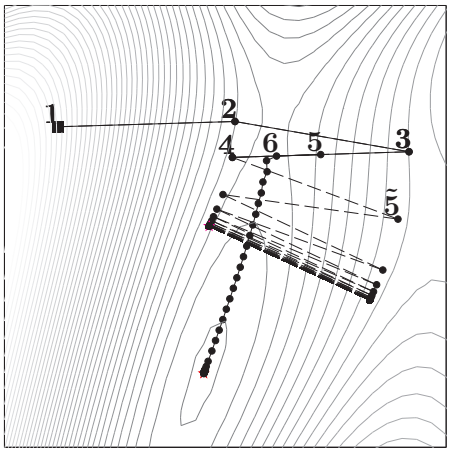

Figure 5: A simple optimization in $d = 2$ dimensions. Symbols and connecting lines mark the trajectories of two gradient based iterations, we display 500 steps for each procedure. Both trajectories start from the same initial position, marked as "1", and employ the same initial step size α_o. Unmodified gradient descent according to Eq. (1) with constant α displays strongly oscillating behavior. The modification with waypoint averaging (here: $k = 2$) and step size adaptation is identical up to step 4, but then replaces the tentative position $\widetilde{5}$ by the mean $(x_3 + x_4)/2$. The step size is then reduced by a factor $r = 1/4$. Also position 6 results from an average over x_4 and x_5, which is very close to the tentative $\widetilde{x}_6$ (not shown). Subsequently the iteration approaches the minimum with step size $\alpha_o/16$ for a number of steps. Close to the minimum many waypoint averages are performed and the step size decreases very rapidly.

be taken into account by normalizing the partial gradients separately and assigning different step sizes to them. In the context of machine learning such subsets could be, for instance, first and second layer weights in a layered neural network. Another example are prototype vectors and relevance matrices in Matrix Relevance LVQ [2]. We employ the latter framework to illustrate an appropriate modification of our method.

As an example data set we consider the *Segmentation* data set as provided by the UCI repository of Machine Learning [8]. The data set contains $(d = 18)$-dim. feature vectors $\mathbf{x}_i$ which are assigned to one of 7 classes denoted by $c(\mathbf{x}_i) \in \{1, 2, \ldots, 7\}$. Note that one of the nominally 19 features does not vary at all and has been omitted here. The training set contains 210 samples (30 per class), 2100 data points (300 per class) serve as a test set. For a more detailed description of the data consult [8] or, for instance, [2].

We consider the simplest setting of GMLVQ with one prototype representing each class. We denote by $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_7]$ a $(7 \cdot 18)$-dim. vector which contains the concatenated prototypes. Classification is parameterized in terms of a nearest prototype

scheme which employs the generalized distance measure

$$d(\mathbf{w_k}, \mathbf{x}) \;=\; (\mathbf{w_k} - \mathbf{x})^{\top}\; \Omega^{\top}\Omega\; (\mathbf{w_k} - \mathbf{x})\,. \tag{7}$$

Here, $\mathbf{x} \in I\!R^d$ represents a feature vector, $\mathbf{w}_k$ is one of the prototypes, and $\Omega \in I\!R^{d\times d}$ is a matrix of adaptive parameters which define the measure.

The training process is guided by the cost function

$$E = \sum_{i=1}^{210} \frac{d(\mathbf{w}_J, \mathbf{x}_i) - d(\mathbf{w}_K, \mathbf{x}_i)}{d(\mathbf{w}_J, \mathbf{x}_i) + d(\mathbf{w}_K, \mathbf{x}_i)} \tag{8}$$

where the sum is over the training examples and the vector $\mathbf{w}_J$ is the prototype representing the class $c(\mathbf{x_i})$. The vector $\mathbf{w}_K$ is the closest prototype representing one of the other classes, as determined according to the distance measure (7).

In GMLVQ the cost function is to be optimized with respect to, both, the prototype positions and the matrix Ω. When applying stochastic gradient descent, it has proven useful to update the elements of Ω with a learning rate different from that for the prototype components [2]. This reflects the fact that the dependence of E on the $\mathbf{w}_j$ and the matrix Ω is expected to be qualitatively different.

In batch descent based on normalized gradients, Eq. (1), we can take this idea into account by performing the normalization for the matrix Ω and the concatenated prototype vector $\mathbf{W}$ separately and using different step sizes in the tentative gradient update corresponding to Eq. (4):

$$\widetilde{\mathbf{W}}_{t+1} \;=\; \mathbf{W}_t \;-\alpha_t^{(W)}\; \frac{\partial E/\partial \mathbf{W}}{|\partial E/\partial \mathbf{W}|} \tag{9}$$

$$\widetilde{\Omega}_{t+1} \;=\; \Omega_t \;-\alpha_t^{(\Omega)}\; \frac{\partial E/\partial \Omega}{|\partial E/\partial \Omega|}. \tag{10}$$

Here we refrain from providing the gradient terms explicitly and refer the reader to [2] for details.

In complete analogy to the above described basic formulation, cf. Eq. 6), the cost function $E(\widetilde{\mathbf{W}}_{t+1}, \widetilde{\Omega}_{t+1})$ is compared with the corresponding costs achieved by

$$\widehat{\mathbf{W}}_{t+1} = \sum_{i=0}^{k-1} \mathbf{W}_{t-i} \;\text{ and }\; \widehat{\Omega}_{t+1} = \sum_{i=0}^{k-1} \Omega_{t-i}.$$

In case the latter is lower, the waypoint average is accepted as the new position and both step sizes are reduced:

$$\alpha_{t+1}^{(W)} = r \cdot \alpha_t^{(W)} \;\text{ and }\; \alpha_{t+1}^{(\Omega)} = r \cdot \alpha_t^{(\Omega)}. \tag{11}$$

As the free parameters of the prescription, one has to set the initial values $\alpha_o^{(W)}$ and $\alpha_o^{(\Omega)}$. Note that their ratio remains fixed in the course of the iteration.

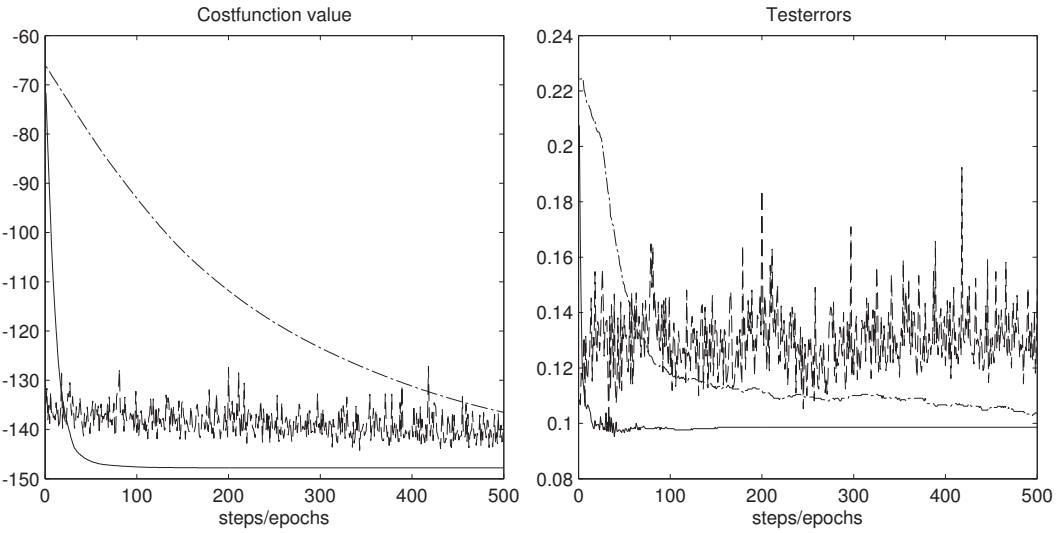

Figure 6: UCI segmentation data set: GMLVQ learning curves for batch gradient descent with constant step sizes (dash–dotted), stochastic gradient descent (dashed), and waypoint averaging with step size adaptation (solid lines). The left panel displays the evolution of the GMLVQ cost function vs. the number of steps (batch methods) or epochs (stochastic descent), respectively. The right panel shows the total classification error with respect to the test set. Parameter settings of the algorithms are specified in the text.

Figure 6 displays learning curves for different variants of gradient based GMLVQ training applied to the UCI segmentation data. We observe only a very weak dependence of the performance on the initial step sizes and on their ratio $\alpha_o^{(\Omega)}/\alpha_o^{(W)}$. The example shown corresponds to $\alpha_o^{(W)} = 1/18$ and $\alpha_o^{(\Omega)} = \alpha_o^{(W)}/2$; the other parameters were $k = 3$ and $r = 2/3$.

For comparison we display example curves for batch gradient descent with constant step sizes $\alpha_o^{(W)} = 1/180$ and $\alpha_o^{(\Omega)} = \alpha_o^{(W)}/5$. These values were chosen such that the outcome after 500 steps is comparable to that of the waypoint averaging procedure with adaptive step size.

Furthermore, display the results of stochastic gradient descent with learning rate schedules of the form

$$\eta(t) = a_1 \, \exp\left[-\ln\left(\frac{a_1}{a_2}\right) \, \frac{t}{t_{max}} \right]$$

where $t_{max=500}$ specifies the maximum number of epochs in the training process. Note that one epoch presents all training examples once and, hence, is to be com-

pared with one step of batch descent. The curves displayed were obtained for $a_1^{(W)} = 0.05, a_2^{(W)} = 0.001$ for the prototype vectors and $a_1^{(\Omega)} = 0.01, a_2^{(\Omega)} = 0.001$ for matrix updates. Note that the stochastic descent displays strong fluctuations which need to be controlled by proper annealing of the learning rate.

Clearly, the performance could be further optimized by choice of the constant step sizes in batch training or the learning rate schedules in the stochastic gradient descent. Potentially, a performance very similar to that of the waypoint averaging procedure could be achieved. However, the important point is that the latter yields very good optimization and classification performance without careful tuning of a number of parameters.

4 Summary and Conclusion

In this Technical Report we present a modification of gradient based optimization which constitutes a conceptually simple extension of steepest descent. The main ingredient is the consideration of *waypoint averages* over the most recent iteration steps in combination with an adaptive step size control. Here we merely present and illustrate the basic concept of the method. We discuss its application in the context of an example machine learning problem: gradient based Matrix Relevance LVQ.

The simple examples considered here already illustrate some of the most attractive features of the method. First of all, it is easy to implement, computationally cheap, and – in contrast to many other schemes – does not require the careful tuning of a large number of algorithm parameters. In particular, the combination of waypoint averages and step size control makes it unnecessary to define explicit learning rate schedules or to use higher derivatives for learning rate adaptation. The use of normalized gradients may appear merely technical at first sight. However, compared to standard steepest descent based on unnormalized gradients, it helps to overcome plateau states and flat regions of the cost function very efficiently.

In a forthcoming publication we will address these aspects in greater depth and demonstrate the flexibility and robustness of the approach in terms of various example problems. A more systematic comparison with alternative, popular methods will also be presented. In addition we will study further mathematical aspects of the approach, including, for instance, the optimal choice of parameters r and k. We will, furthermore, demonstrate that the method is suitable also in situations in which the cost function is not differentiable in the minimum.

A number of question deserves particular attention in the context of machine learning, e.g. the convergence behavior in the presence of extended plateaus or many local minima.

References

[1] Wei Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *CoRR*, abs/1107.2490, 2011.

[2] P. Schneider, M. Biehl, and B. Hammer. Adaptive relevance matrices in learning vector quantization. *Neural Computation*, 21(12):3532–3561, 2009.

[3] T. Villmann, B. Hammer, and M. Biehl. Some theoretical aspects of the neural gas vector quantizer. In M. Biehl, B. Hammer, M. Verleysen, and T. Villmann, editors, *Similarity-Based Clustering*, volume 5400, pages 23–34. Springer Lecture Notes in Computer Science, 2009.

[4] A. Witoelar, M. Biehl, A. Ghosh, and B. Hammer. Learning dynamics and robustness of vector quantization and neural gas. *Neurocomputing*, 71(7-9):1210–1219, 2008.

[5] M.A. Arbib, editor. *The handbook of brain theory and neural networks*. MIT Press, Cambridge,MA, 2003.

[6] M. Biehl and N. Caticha. The statistical mechanics of on-line learning and generalization. *The handbook of brain theory and neural networks*, pages 1095–1098, 2003.

[7] D. Saad, editor. *Online learning in neural networks*. Cambridge University Press, Cambridge, UK, 1999.

[8] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases. `http://archive.ics.uci.edu/ml/`, 1998.

[9] M. Biehl, A. Freking, and G. Reents. Dynamics of on-line competitive learning. *Europhysics Letters*, 38:73–78, 1997.

[10] M. Biehl, P. Riegler, and C. Wöhler. Transient dynamics of on-line learning in two-layered neural networks. *Journal of Physics A: Mathematical and General*, 29:4769–4780, 1996.

[11] A. Sato and K. Yamada. Generalized learning vector quantization. In M. C. Mozer D. S. Touretzky and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8. Proceedings of the 1995 Conference*, pages 423–9, Cambridge, MA, USA, 1996. MIT Press.

[12] Y. Chauvin and D.E. Rumelhart, editors. *Backpropagation: Theory, Architectures, and Applications*. Hillsdale, Erlbaum, 1995.

[13] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1 edition, 1995.

[14] D. Saad and S.A. Solla. Exact Solution for Online Learning in Multilayer Neural Networks. *Phys. Rev. Lett.*, 74:4337–4340, 1995.

[15] T. Martinetz, S. Berkovich, and K. Schulten. Neural gas network for vector quantization and its application to time series prediction. *IEEE Trans. Neural Networks*, 4:558–569, 1993.

[16] B.T. Polyak and A.B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control and Optimization*, 30:838–855, 1992.

[17] J.A. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation*. Addison Wesley, Redwood City, 1991.

[18] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, 2nd edition, 1987.

[19] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing*, volume 1. Bradford Books, Cambridge and London, 1987.

[20] H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22:400–407, 1951.