Stochastic Learning

Léon Bottou

NEC Labs of America, 4 Independence Way, Princeton NJ08540, USA
leonb@bottou.org
WWW home page: http://leon.bottou.org

Abstract. This contribution presents an overview of the theoretical and practical aspects of the broad family of learning algorithms based on Stochastic Gradient Descent, including Perceptrons, Adalines, K-Means, LVQ, Multi-Layer Networks, and Graph Transformer Networks.

1 Introduction

This contribution reviews some material presented during the "Stochastic Learning" lecture given at the 2003 Machine Learning Summer School in Tübingen. It defines a broad family of learning algorithms that can be formalized as stochastic gradient descent algorithms and describes their common properties. This includes numerous well known algorithms such as Perceptrons, Adalines, K-Means, LVQ, and Multi-Layer Networks.

Stochastic learning algorithms are also effective for training large systems with rich structure, such as Graph Transformer Networks [8, 24]. Such large scale systems have been designed and industrially deployed with considerable success.

- Section 2 presents the basic framework and illustrates it with a number of well known learning algorithms.
- Section 3 presents the basic mathematical tools for establishing the convergence of stochastic learning algorithms.
- Section 4 discusses the learning speed of stochastic learning algorithms applied to large datasets. This discussion covers both statistical efficiency and computational requirements.

These concepts were previously discussed in [9, 10, 14, 12]. Readers interested by the practice of stochastic gradient algorithms should also read [25] and investigate applied contributions such as [39, 37, 46, 6, 24, 26].

2 Foundations

Almost all of the early work on *Learning Systems* focused on online algorithms [18, 34, 44, 2, 19]. In these early days, the algorithmic simplicity of online algorithms was a requirement. This is still the case when it comes to handling large, real-life training sets [23, 30, 25, 26].

The early *Recursive Adaptive Algorithms* were introduced during the same years [33] and very often by the same people [45]. First developed in the engineering world, recursive adaptation algorithms have turned into a mathematical discipline, namely *Stochastic Approximations* [22, 27, 7].

2.1 Expected Risk Function

In [40, 41], the goal of a learning system consists of finding the minimum of a function $C(w)$ named the *expected risk function*[1]. This function is decomposed as follows:

$$C(w) \;\stackrel{\triangle}{=}\; \mathbf{E}_z\, Q(z,w) \;\stackrel{\triangle}{=}\; \int Q(z,w)\; dP(z) \tag{1}$$

The minimization variable w is meant to represent the part of the learning system which must be adapted as a response to observing events z occurring in the real world. The *loss function* $Q(z,w)$ measures the performance of the learning system with parameter w under the circumstances described by event z. Common mathematical practice suggests to represent both w and z by elements of adequately chosen spaces $\mathcal{W}$ and $\mathcal{Z}$.

2.2 Gradient Based Learning

The expected risk function (1) cannot be minimized directly because the grand truth distribution is unknown. It is however possible to compute an approximation of $C(w)$ by simply using a finite *training set* of independent observations $z_1, \ldots, z_L$.

$$C(w) \;\approx\; \hat{C}_L(w) \;\stackrel{\triangle}{=}\; \frac{1}{L}\sum_{n=1}^{L} Q(z_n,w) \tag{2}$$

General theorems [42] show that minimizing the *empirical risk* $\hat{C}_L(w)$ can provide a good estimate of the minimum of the expected risk $C(w)$ when the training set is large enough. This line of work has provided a way to understand the *generalization* phenomenon, i.e. the ability of a system to learn from a finite training set and yet provide results that are valid in general.

Batch Gradient Descent. Minimizing the empirical risk $\hat{C}_L(w)$ can be achieved using a *batch gradient descent* algorithm. Successive estimates w_t of the optimal parameter are computed using the following formula

$$w_{t+1} \;=\; w_t - \gamma_t \nabla_w\, \hat{C}_L(w_t) \;=\; w_t - \gamma_t \frac{1}{L}\sum_{i=1}^{L} \nabla_w\, Q(z_n,w_t) \tag{3}$$

[1] The origin of this statistical framework is unclear. It has been popularized by Vapnik's work [42] but was already discussed in Tsypkin's work [40] or even [16]. Vapnik told me that "someone wrote this on the blackboard during a seminar"; he does not remember who did.

where the learning rate γ_t is a positive number.

The properties of this optimization algorithm are well known: When the learning rate γ_t are small enough[2], the algorithm converges towards a local minimum of the empirical risk $\hat{C}_L(w)$. Each iteration of the batch gradient descent algorithm however involves a burdening computation of the average of the gradients of the loss function $\nabla_w Q(z_n, w)$ over the entire training set. Significant computer resources must be allocated in order to store a large enough training set and compute this average.

Online Gradient Descent. The elementary *online gradient descent* algorithm is obtained by dropping the averaging operation in the batch gradient descent algorithm (3). Instead of averaging the gradient of the loss over the complete training set, each iteration of the online gradient descent consists of choosing an example z_t at random, and updating the parameter w_t according to the following formula.

$$w_{t+1} \;=\; w_t - \gamma_t \nabla_w Q(z_t, w_t) \tag{4}$$

Averaging this update over all possible choices of the training example z_t would restore the batch gradient descent algorithm. The online gradient descent simplification relies on the hope that the random noise introduced by this procedure will not perturbate the average behavior of the algorithm. Significant empirical evidence substantiate this hope.

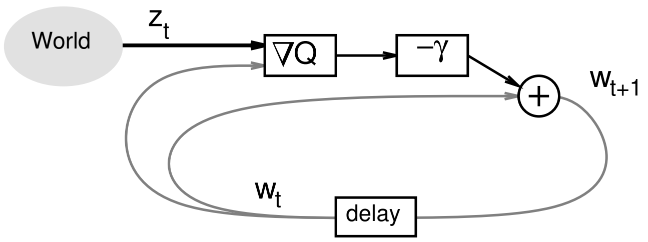


Fig. 1. : Online Gradient Descent. The parameters of the learning system are updated using information extracted from real world observations.

Many variants of (4) have been defined. Parts of this contribution discuss two significant variants: Section 2.4 replaces the gradient $\nabla_w Q(z, w)$ by a general term $U(z, w)$ satisfying $\mathbf{E}_z\, U(z, w) = \nabla_w C(w)$. Section 4 replaces the learning rates γ_t by positive symmetric matrices (equation (27).)

Online gradient descent can also be described without reference to a training set. Instead of drawing examples from a training set, we can directly use the events z_t observed in the real world, as shown in Figure 1. This formulation

[2] Convergence occurs for constant learning rates, smaller than a critical learning rate related to the maximal curvature of the cost function. See [25] for instance.

is particularly adequate for describing *adaptive algorithms* that simultaneously process an observation and learn to perform better. Such adaptive algorithms are very useful for tracking a phenomenon that evolves in time.

Formulating online gradient descent without reference to a training set also presents a theoretical interest. Each iteration of the algorithm uses an example z_t drawn from the grand truth distribution instead of a finite training set. The average update therefore is a gradient descent algorithm which directly optimizes the expected risk. This shortcuts the usual discussion about differences between optimizing the empirical risk and the expected risk [42, 43]. Proving the convergence of an online algorithm towards the minimum of the expected risk provides an alternative to the Vapnik proofs of the consistency of learning algorithms. Non-asymptotic bounds for online algorithms are rare.

2.3 Examples: Online Least Mean Squares

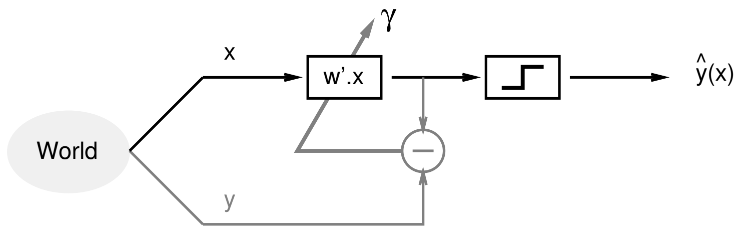


Fig. 2. : Widrow's Adaline. The adaline computes a binary indicator by thresholding a linear combination of its input. Learning is achieved using the *delta rule*.

Widrow's Adaline. The *Adaline* [44] is one of the few learning systems designed at the very beginning of the computer age. Online gradient descent was then a very attractive proposition requiring little hardware. The adaline could fit in a refrigerator sized cabinet containing a forest of potentiometers and electrical motors.

The Adaline (Figure 2) learning algorithm adapts the parameters of a single *threshold unit*. Input patterns x are recognized as class $y = +1$ or $y = -1$ according to the sign of $w'x + \beta$. It is practical to consider an *augmented input* pattern x containing an extra constant coefficient equal to 1. The bias β then is represented as an extra coefficient in the parameter vector w. With this convention, the output of the threshold unit can be written as

$$\hat{y}_w(x) \;\triangleq\; \mathrm{sign}(w'x) \;=\; \mathrm{sign}\sum_i w_i x_i \tag{5}$$

During training, the Adaline is provided with pairs $z = (x, y)$ representing input patterns and desired output for the Adaline. The parameter w is adjusted after using the *delta rule* (the "prime" denotes transposed vectors):

$$w_{t+1} = w_t - \gamma_t (y_t - w_t' x_t)' x_t \tag{6}$$

This delta rule is nothing more than an iteration of the online gradient descent algorithm (4) with the following loss function:

$$Q_{\text{adaline}}(z, w) \ \stackrel{\triangle}{=}\ (y - w'x)^2 \tag{7}$$

This loss function does not take the discontinuity of the threshold unit (5) into account. This linear approximation is a real breakthrough over the apparently more natural loss function $(y - \hat{y}_w(x))^2$. This discontinuous loss function is difficult to minimize because its gradient is zero almost everywhere. Furthermore, all solutions achieving the same misclassification rate would have the same cost $C(w)$, regardless of the margins separating the examples from the decision boundary implemented by the threshold unit.

Multi-Layer Networks. *Multi-Layer Networks* were initially designed to overcome the computational limitation of the threshold units [29]. Arbitrary binary mappings can be implemented by stacking several layers of threshold units, each layer using the outputs of the previous layers as inputs. The Adaline linear approximation could not be used in this framework, because ignoring the discontinuities would make the entire system linear regardless of the number of layers. The key of a learning algorithm for multi-layer networks [35] consisted of noticing that the discontinuity of the threshold unit could be represented by a smooth non-linear approximation.

$$\text{sign}(w'x) \ \approx\ \tanh(w'x) \tag{8}$$

Using such *sigmoid units* does not reduce the computational capabilities of a multi-layer network, because the approximation of a step function by a sigmoid can be made arbitrarily good by scaling the coefficients of the parameter vector w.

A multi-layer network of sigmoidal units implements a differentiable function $f(x, w)$ of the input pattern x and the parameters w. Given an input pattern x and the desired network output y, the *back-propagation* algorithm, [35] provides an efficient way to compute the gradients of the mean square loss function.

$$Q_{\text{mse}}(z, w) = \frac{1}{2}\,(y - f(x, w))^2 \tag{9}$$

Both the batch gradient descent (3) and the online gradient descent (4) have been used with considerable success. On large, redundant data sets, the online version converges much faster then the batch version, sometimes by orders of magnitude [30]. An intuitive explanation can be found in the following extreme example. Consider a training set composed of two copies of the same subset. The batch algorithm (3) averages the gradient of the loss function over the whole training set, causing redundant computations. On the other hand, running online gradient descent (4) on all examples of the training set would amount to performing two complete learning iterations over the duplicated subset.

2.4 Examples: Non Differentiable Loss Functions

Many interesting examples involve a loss function $Q(z, w)$ which is not differentiable on a subset of points with probability zero. Intuition suggests that this is a minor problems because the iterations of the online gradient descent have zero probability to reach one of these points. Even if we reach one of these points, we can just draw another example z.

This can be formalized as replacing the gradient $\nabla_w Q(z, w)$ in equation (4) by an update term $U(z, w)$ defined as follows:

$$U(z,w) = \begin{cases} \nabla_w Q(z,w) & \text{when differentiable} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The convergence study (Section 3) shows that this works if the expectation of the update term $U(z, w)$ is equal to gradient of the cost $C(w)$:

$$\begin{aligned} \mathbf{E}_z U(z,w) &\stackrel{?}{=} \nabla_w C(w) \\ \int \nabla_w Q(z,w)\, dP(z) &\stackrel{?}{=} \nabla_w \int Q(z,w)\, dP(z) \end{aligned} \tag{11}$$

The Lebesgue integration theory provides a sufficient condition for swapping the integration ($\int$) and differentiation (∇_w) operators as in (11). For each parameter value w reached by the online algorithm, it is sufficient to find an integrable function $\Phi(z, w)$ and a neighborhood $\vartheta(w)$ of w such that:

$$\forall z,\ \forall v \in \vartheta(w),\quad |Q(z,v) - Q(z,w)| \ \le\ |w - v|\, \Phi(z,w) \tag{12}$$

This condition (12) tests that the maximal slope of the loss function $Q(z, w)$ is conveniently bounded. This is obviously true when the loss function $Q(z, w)$ is differentiable and has an integrable gradient. This is obviously false when the loss function is not continuous. Given our previous assumption concerning the zero probability of the non differentiable points, condition (12) is a sufficient condition for safely ignoring a few non differentiable points.

Rosenblatt's Perceptron. During the early days of the computer age, the *Perceptron* [34] generated considerable interest as a possible architecture for general purpose computers. This interest faded after the disclosure of its computational limitations [29]. Figure 3 represents the perceptron architecture. An *associative area* produces a feature vector x by applying predefined transformations to the *retina* input. The feature vector is then processed by a *threshold unit* (cd. Adaline).

The perceptron learning algorithm adapts the parameters w of the threshold unit. Whenever a misclassification occurs, the parameters are updated according to the *perceptron rule*.

$$w_{t+1} = w_t + 2\gamma_t y_t\, x_t \tag{13}$$

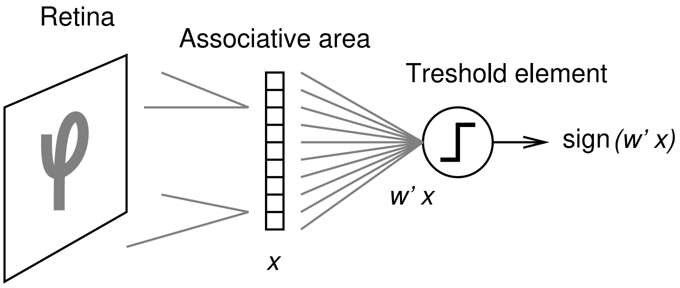


Fig. 3. : Rosenblatt's Perceptron is composed of a fixed preprocessing and of a trainable threshold unit.

This learning rule can be derived as an online gradient descent applied to the following loss function:

$$Q_{\mathrm{perceptron}}(z, w) = (\mathrm{sign}(w'x) - y)\, w'x \tag{14}$$

Although this loss function is non differentiable when $w'x$ is null, is meets condition (12) as soon as the expectation $\mathbf{E}(x)$ is defined. We can therefore ignore the non differentiability and apply the online gradient descent algorithm:

$$w_{t+1} = w_t - \gamma_t(\mathrm{sign}(w_t' x_t) - y_t)\, x_t \tag{15}$$

Since the desired class is either $+1$ or -1, the weights are not modified when the pattern x is correctly classified. Therefore this parameter update (15) is equivalent to the perceptron rule (13).

The perceptron loss function (14) is zero when the pattern x is correctly recognized as a member of class $y = \pm 1$. Otherwise its value is positive and proportional to the dot product $w'x$. The corresponding cost function reaches its minimal value zero when all examples are properly recognized or when the weight vector w is null.

Such *hinge loss functions* [17, 36] have recently drawn much interest because of their links with the Support Vector Machines and the AdaBoost algorithm.

***K*-Means.** The *K-Means* algorithm [28] is a popular clustering method which dispatches K centroids $w_{(k)}$ in order to find clusters in a set of points $x_1, \ldots, x_L$. This algorithm can be derived by performing the online gradient descent with the following loss function.

$$Q_{\mathrm{kmeans}}(x, w) \;\stackrel{\triangle}{=}\; \min_{k=1}^{K}\, (x - w_{(k)})^2 \tag{16}$$

This loss function measures the quantification error, that is to say the error on the position of point x when we replace it by the closest centroid. The corresponding cost function measures the average quantification error.

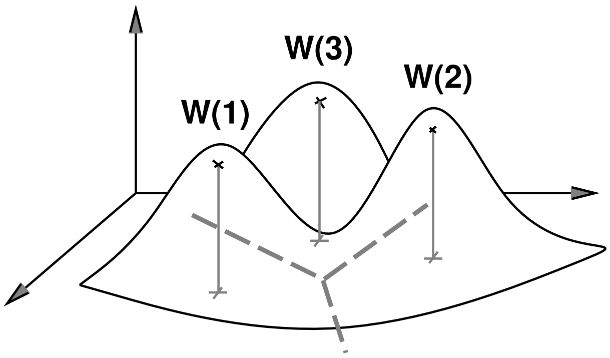


Fig. 4. : K-Means dispatches a predefined number of cluster centroids in a way that minimizes the quantification error.

This loss function is not differentiable on points located on the Voronoï boundaries of the set of centroids, but meets condition (12) as soon as the expectations $\mathbf{E}(x)$ and $\mathbf{E}(x^2)$ are defined. On the remaining points, the derivative of the loss is the derivative of the distance to the nearest centroid w^-. We can therefore ignore the non-differentiable points and apply the online gradient descent algorithm.

$$w_{t+1}^- = w_t^- + \gamma_t (x_t - w_t^-) \tag{17}$$

This formula describes an elementary iteration of the K-Means algorithm. A very efficient choice of learning rates γ_t will be suggested in Section 4.6.

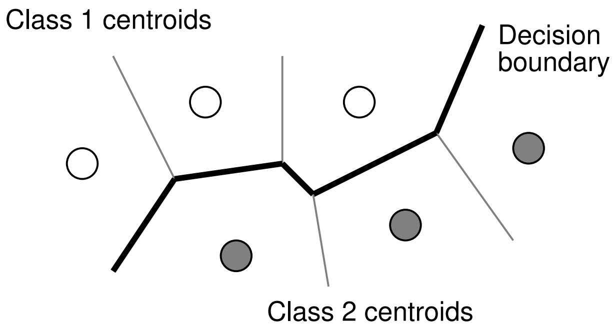


Fig. 5. : Kohonen's LVQ2 pattern recognition scheme outputs the class associated with the closest reference point to the input pattern.

Learning Vector Quantization 2. Kohonen's *Learning Vector Quantization 2* (LVQ2) rule [20] is a powerful pattern recognition algorithm. Like K-Means, it uses a fixed set of reference points $w(k)$. A class $y(k)$ is associated with each reference point. As shown in Figure 5, an unknown pattern x is then recognized as a member of the class associated with the nearest reference point.

Given a training pattern x, let us denote w^- the nearest reference point and denote w^+ the nearest reference point among those associated with the correct class y. Adaptation only occurs when the closest reference point w^- is associated with an incorrect class while the closest correct reference point w^+ is not too far away:

$$\text{if} \begin{cases} x \text{ is misclassified } (w^- \neq w^+) \\ \text{and } \ (x - w^+)^2 < (1+\delta)(x - w^-)^2 \end{cases} \qquad \text{then} \begin{cases} w^-_{t+1} = w^-_t - \varepsilon_t (x - w^-_t) \\ w^+_{t+1} = w^+_t + \varepsilon_t (x - w^+_t) \end{cases} \tag{18}$$

Reference points are only updated when the pattern x is misclassified. Furthermore, the distance to the closest correct reference point w^+ must not exceed the distance to the closest (incorrect) reference point w^- by more than a percentage defined by parameter δ. When both conditions are met, the algorithm pushes the closest (incorrect) reference point w^- away from the pattern x, and pulls the closest correct reference point w^+ towards the pattern x.

This intuitive algorithm can be derived by performing an online gradient descent with the following loss function:

$$Q_{\text{lvq2}}(z, w) \ \triangleq \ \begin{cases} 0 & \text{if } x \text{ is well classified } (w^+ = w^-) \\ 1 & \text{if } (x - w^+)^2 \geq (1+\delta)(x - w^-)^2 \\ \frac{(x-w^+)^2 - (x-w^-)^2}{\delta (x-w^-)^2} & \text{otherwise} \end{cases} \tag{19}$$

This function is a continuous approximation to a binary variable indicating whether pattern x is misclassified. The corresponding cost function therefore is a continuous approximation of the system misclassification rate [9]. This analysis helps understanding how the LVQ2 algorithm works.

Although the above loss function is not differentiable for some values of w, it meets condition (12) as soon as the expectations $\mathbf{E}(x)$ and $\mathbf{E}(x^2)$ are defined. We can therefore ignore the non-differentiable points and apply the online gradient descent algorithm:

$$\text{if} \begin{cases} x \text{ is misclassified } (w^- \neq w^+) \\ \text{and } \ (x - w^+)^2 < (1+\delta)(x - w^-)^2 \end{cases} \qquad \text{then} \begin{cases} w^-_{t+1} = w^-_t - \gamma_t k_1 (x - w^-_t) \\ w^+_{t+1} = w^+_t + \gamma_t k_2 (x - w^+_t) \end{cases} \tag{20}$$

$$\text{with } \ k_2 = \frac{1}{\delta (x - w^-)^2} \ \text{ and } \ k_1 = k_2 \frac{(x - w^+)^2}{(x - w^-)^2} \tag{21}$$

This online gradient descent algorithm (20) is similar to the usual LVQ2 learning algorithm (18). The difference between the two scalar coefficients k_1 and k_2 can be viewed as a minor variation on the learning rates.

3 Convergence

Given a suitable choice of the learning rates γ_t, the batch gradient descent algorithm (3) is known to converge to a local minimum of the cost function. This local minimum is a function of the initial parameters w_0. The parameter trajectory follows the meanders of the local attraction basin and eventually reaches the corresponding minimum.

The random noise introduced by stochastic gradient descent (4) disrupts this deterministic picture. The parameter trajectory can jump from basin to basin. One usually distinguish a *search phase* that explores the parameter space and a *final phase* that takes place in the vicinity of a minimum.

– The final phase takes place in the vicinity of a single local minimum w^* where the cost function is essentially convex. This is discussed in Section 3.1.
– Our understanding of the search phase is still very spotty. Section 3.2 presents sufficient conditions to guarantee that the convergence process will eventually reach the final phase.

3.1 Final convergence phase

The following discussion rely on the *general convexity assumption*[3]. Everywhere in the parameter space, the opposite of the gradient must point toward a unique minimum w^*.

$$\forall \varepsilon > 0, \qquad \inf_{(w-w^*)^2 > \varepsilon} (w - w^*)\, \nabla_w C(w) > 0 \tag{22}$$

Such a strong assumption is only valid for a few simple learning algorithms such as the Adaline, Section 2.3). Nevertheless these results are useful for understanding the final convergence phase. The assumption usually holds within the final convergence region because the cost function is locally convex.

The parameter updates $\gamma_t \nabla_w Q(z, w)$ must become smaller and smaller when the parameter vector w approaches the optimum w^*. This implies that either the gradients or the learning rates must vanish in the vicinity of the optimum.

More specifically one can write:

$$\mathbf{E}_z \left[\nabla_w Q(z,w)^2 \right] = \mathbf{E}_z \left[\left(\nabla_w Q(z,w) - \nabla_w C(w) \right)^2 \right] + \left\| \nabla_w C(w) \right\|^2$$

The first term is the variance of the stochastic gradient. It is reasonable to assume that it does not grow faster than the norm of the real gradient itself. In the vicinity of w^* we can write:

$$\left\| \nabla_w C(w) \right\|^2 = \left\| \nabla_w C(w) - \nabla_w C(w^*) \right\|^2 \leq \frac{1}{2} \left\| \nabla \nabla_w C(w^*) \right\|^2 \; \left\| w - w^* \right\|^2$$

[3] The optimization literature often defines such extended notions of convexity. Small details are important. For instance, in (22), one cannot simply replace the infimum by $\forall w \neq w^*$. Consider function $C(w) = 1 - \exp(-||w||^2)$ as a counter-example.

It is therefore reasonable to assume that $||\nabla_w C(w)||^2$ behaves quadratically within the final convergence region. Both assumptions are conveniently expressed as follows:

$$\mathbf{E}_z \left[\nabla_w Q(z,w)^2 \right] < A + B\,(w - w^*)^2 \quad \text{with } A \geq 0,\ B \geq 0 \tag{23}$$

The constant A must be greater than the residual variance $\mathbf{E}_z \left[\nabla_w Q(z,w^*)^2 \right]$ of the gradients at the optimum. This residual variance can be zero for certain rare noiseless problems where w^* simultaneously optimizes the loss for every examples. It is strictly positive in most practical cases. The average norm of the gradients then does not vanish when the parameter vector w approaches the optimum. Therefore one must use *decreasing learning rates*, e.g.:

$$\sum \gamma_t^2 < \infty \tag{24}$$

The presence of constant A in (23) marks a critical difference between stochastic and ordinary gradient descent. There is no such constant in the case of the ordinary gradient descent. A simple analysis then yields an expression for the maximal constant learning rate [25]. In the stochastic gradient case, this analysis suggests that the parameter vector eventually hovers around the minimum w^* at a distance roughly proportional to γ_t. Quickly decreasing the learning rate is therefore tempting. Suppose however that the learning rates decrease so fast that $\sum \gamma_t = R < \infty$. This would effectively maintain the parameters within a certain radius of their initial value. It is therefore necessary to enforce the following condition:

$$\sum \gamma_t = \infty \tag{25}$$

Convex convergence theorem. *The general convexity (22) and the three conditions (23), (24) and (25) are sufficient conditions for the almost sure convergence of the stochastic gradient descent (4) to the optimum w^*.*

The following discussion provides a sketch of the proof. This proof is simply an extension of the convergence proofs for the continuous gradient descent and the discrete gradient descent.

The continuous gradient descent proof studies the convergence of the function $w(t)$ defined by the following differential equation:

$$\frac{\mathrm{d}w}{\mathrm{d}t} = -\nabla_w C(w)$$

This proof follows three steps:

A) *Definition of a Lyapunov function* — A Lyapunov function is a function whose convergence to zero implies the convergence of $w(t)$ to w^* when t grows to the infinity. For the continuous gradient we simply use $h(t) = (w - w^*)^2$.

B) *Convergence of the Lyapunov function* — Using (22), it is easy to see that $\mathrm{d}h/\mathrm{d}t = 2(w - w^*)\nabla_w C(w) \leq 0$. Function $h(t)$ converges because it is both positive and decreasing.

C) *The limit of the Lyapunov function must be zero.* We know that $\mathrm{d}h/\mathrm{d}t \to 0$ because $h(t)$ converges. Assumption (22) then implies that $(w - w^*)^2 \to 0$.

The convergence proofs for both the discrete (3) and stochastic (4) gradient descents follow the same three steps. Each step requires increasingly sophisticated mathematical tools summarized in the following table.

	Continuous	**Discrete**	**Stochastic**
Step A Define Lyapunov criterion.	Function $h(t) = (w(t) - w^*)^2$	Sequence $h_t = (w_t - w^*)^2$	Random Process $h_t = (w_t - w^*)^2$
Step B Lyapunov criterion converges.	Decreasing positive function	Positive sequence with bounded positive increments	Positive quasi-martingales
Step C Lyapunov criterion converges to zero.	General Convexity		

Full details can be found in [9, 10].

3.2 Search phase

This section discusses the convergence of the stochastic gradient algorithm (4) without the general convexity assumption (22). Since the cost function $C(w)$ can have several local minima, this discussion encompasses the search phase. Although our understanding of the search phase is still very incomplete, empirical and theoretical evidence indicate that stochastic gradient algorithms enjoy significant advantages over batch algorithms. Stochastic gradient descent benefit from the redundancies of the training set. Consider the extreme case where a training set of size 1000 is inadvertently composed of 10 identical copies of a set with 100 samples. Averaging the gradient over all 1000 patterns gives the exact same result as computing the gradient based on just the first 100. Batch gradient descent is wasteful because it re-computes the same quantity 10 times before one parameter update. On the other hand, stochastic gradient will see a full epoch as 10 iterations through a 100-long training set.

In the case of the continuous and discrete gradient descent, it is usually sufficient to partition the parameter space into several attraction basins, discuss the conditions under which the algorithm confines the parameters w_t in a single attraction basin, define a suitable Lyapunov criterion [21], and proceed as in the convex case. This approach does not work well with stochastic gradient because the parameter trajectory can always jump from basin to basin.

Let us instead assume that the cost function becomes large when one wanders far from the origin. The global landscape then looks like a single large attraction basin. The local minima structure only shows when one gives a closer look to in the vicinity of the apparent minimum.

This situation can be expressed by the following assumptions:

$$
\begin{aligned}
i.)&\ \inf C(w) > -\infty \\
ii.)&\ \exists D > 0,\ \inf_{w^2 > D} w\,\nabla_w C(w) > 0 \\
iii.)&\ \mathbf{E}_z(\nabla_w Q(z,w))^2 \leq A + Bw^2 \\
iv.)&\ \exists E > D,\ \forall z,\ \sup_{w^2 < E} ||\nabla_w Q(z,w)|| \leq Constant
\end{aligned}
$$

Assumption (i) indicates that the cost is bounded from below. Assumption (ii) indicates that the gradient far away from the origin always drives us back towards the origin. Assumptions (iii) and (iv) limit the variance of the stochastic gradient and the asymptotic growth of the real gradients[4].

Global confinement theorem: *The four assumptions* (i) *to* (iv) *above, and the two learning rate assumptions (24) and (25) guarantee that the parameter* w_t *defined by the stochastic gradient update (4) will almost surely remain confined within distance* $\sqrt{E}$ *of the origin.*

This global confinement result [10] is obtained using the same proof technique as in the convex case. The Lyapunov criterion is simply defined as $h_t = \max(E, w_t^2)$.

Global confinement shows that w_t evolves in a compact domain where nothing dramatic can happen. In fact, it even implies that the stochastic gradient descent will soon or later reach the final convergence phase. This is formalized by the following result:

Gradient convergence theorem. *The four assumptions* (i) *to* (iv) *above, and the two learning rate assumptions (24) and (25) guarantee that the gradients* $\nabla_w C(w_t)$ *converges almost surely to zero.*

The proof of this final convergence result [10] again is very similar to the convergence proofs for the convex case with suitable choices for the Lyapunov criterion. The details of the proof extensively rely on the global confinement result.

4 Convergence speed and learning speed

The main purpose of this section is to illustrate a critical difference between optimization algorithms and learning algorithm. It will then appear that stochastic gradient descent is simultaneously a very poor optimization algorithm and a very effective learning algorithm.

4.1 Convergence speed for batch gradient descent

Simple batch gradient descent enjoy *linear*[5] convergence speed (see for instance Section 5 of [25]). The convergence speed of batch gradient descent drastically

[4] See also the discussion for convex assumption (23).

[5] Linear convergence speed: $(\log 1/(w_t - w^*)^2)$ grows linearly with t.

improves when one replaces the scalar learning rates γ_t by a definite positive symmetric matrix Φ_t that approximates the inverse of the Hessian of the cost function.

$$\Phi_t \approx H^{-1}(w_t), \qquad H(w) = \nabla\nabla_w C(w) \tag{26}$$

This leads to very effective optimization algorithms such as Newton's algorithm, Levemberg-Marquardt, Conjugate Gradient and BFGS (see [15] for a review). These algorithms achieve *superlinear* or even *quadratic*[6] convergence speeds.

4.2 Convergence speed for stochastic algorithms

Whereas online algorithms may converge to the general area of the optimum at least as fast as batch algorithms [25], the optimization proceeds rather slowly during the final convergence phase [14]. The noisy gradient estimate causes the parameter vector to fluctuate around the optimum in a bowl whose size depends on the decreasing learning rates and is therefore constrained by (25). It can be shown that this size decreases like $1/t$ at best[7].

Stochastic gradient descent nevertheless benefits from using similar second order methods. The gradient vector is rescaled using a positive symmetric matrix Φ_t that approximates the inverse hessian (26) in a manner analogous to Newton's algorithm[8]. The same convergence results apply as long as the eigenvalues of the scaling matrix Φ_t are bounded.

$$w_{t+1} \;=\; w_t - \frac{1}{t}\,\Phi_t\,\nabla_w\,Q(z_t, w_t) \tag{27}$$

For simplicity, this section only addresses the case $\gamma_t = 1/t$ which satisfies both conditions (24) and (25). It is however important to understand that the second order stochastic algorithm (27) still experiences the stochastic noise resulting from the random selection of the examples z_t. Its convergence speed still depends on the choice of decreasing learning rates γ_t and is therefore constrained by condition (25). This is a sharp contrast with the case of batch algorithms where the same scaling matrix yields superlinear convergence.

Stochastic gradient descent is a hopeless optimization algorithm. It is tempting to conclude that it is also a very poor learning algorithm. Yet experience suggests otherwise [4].

4.3 Optimization versus Learning

This apparent contradiction is resolved when one considers that the above discussion compares the speed of two different convergences:

- Batch algorithms converge towards a minimum of the *empirical risk* $\hat{C}_L(w)$, which is defined as an average on L training examples (2).

[6] Quadratic convergence speed: $(\log\log 1/(w_t - w^*)^2)$ grows linearly with t.
[7] Convergence speed of stochastic gradient: $(1/(w_t - w^*)^2)$ grows linearly with t.
[8] Such second order stochastic approximations are standard practice in the Stochastic Approximation literature [22, 27, 7].

– Stochastic algorithms converge towards a minimum of the *expected risk* $C(w)$, which is defined as an expectation with respect to the probability distribution from which we draw the examples (1).

In a learning problem, we are interested in knowing the speed of convergence towards the minimum of the expected risk $C(w)$ because it reflects the generalization error. Replacing the expected risk $C(w)$ by the empirical risk $\hat{C}_L(w)$ is by itself an approximation. As shown in the next section, this approximation spoils the potential benefits of running an optimization algorithm with ambitious convergence speed.

4.4 Optimizing the empirical risk is a stochastic process

We consider in this section an infinite sequence of independent training examples $(z_1, \dots, z_t, \dots)$. Let w_t^* be the minimum of the empirical risk $\hat{C}_t(w)$ defined on a training set composed of the first t examples $(z_1, \dots, z_t)$. We assume that all the w_t^* are located in the vicinity of the minimum w^* of the expected risk $C(w)$.

Manipulating a Taylor expansion of the gradient of $\hat{C}_{t+1}(w)$ in the vicinity of w_t^* provides the following recursive relation:

$$w_{t+1}^* \;=\; w_t^* - \frac{1}{t+1}\Psi_t \nabla_w Q(z_t, w_t^*) + \mathcal{O}\left(\frac{1}{t^2}\right) \tag{28}$$

with

$$\Psi_t \;\triangleq\; \left(\frac{1}{t+1}\sum_{i=1}^{t+1} \nabla\nabla_w Q(z_i, w_t^*)\right)^{-1} \underset{t\to\infty}{\longrightarrow} H^{-1}(w_t^*)$$

The similarity between (28) and (27) suggests that both the batch sequence (w_t^*) and online sequence (w_t) converge at the same speed for adequate choices of the scaling matrix Φ_t. Theoretical analysis indeed shows that [31, 13]:

$$\mathbf{E}\left[\,(w_t^* - w^*)^2\,\right] = \frac{K}{t} + \mathrm{o}\left(\frac{1}{t}\right) \tag{29}$$

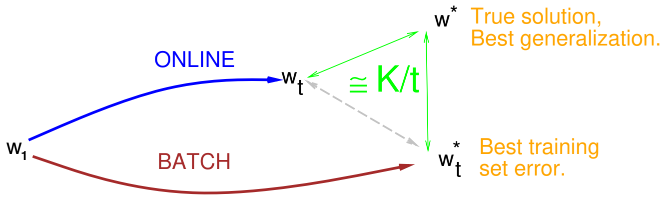
$$\Phi_t \underset{t\to\infty}{\longrightarrow} H^{-1}(w^*) \;\;\Longrightarrow\;\; \mathbf{E}\left[\,(w_t - w^*)^2\,\right] = \frac{K}{t} + \mathrm{o}\left(\frac{1}{t}\right) \tag{30}$$

where

$$K = \mathrm{trace}\left(H^{-1}(w^*)\,\cdot\,\mathbf{E}_z\left[(\nabla_w Q(z,w^*))\,(\nabla_w Q(z,w^*))'\right]\,\cdot\,H^{-1}(w^*)\right)$$

Not only does this result establish that both sequences have $\mathcal{O}(1/t)$ convergence, but also it provides the value of the common constant K. This constant is neither affected by the second order terms of (28) nor by the convergence speed of the scaling matrix Φ_t towards the inverse Hessian [13].

Following [40], we say that a second order stochastic algorithm is *optimal* when Φ_t converges to $H^{-1}(w^*)$. Figure 6 summarizes the behavior of such optimal algorithms. After t iterations on fresh examples, the point w_t reached by an optimal stochastic learning algorithm is asymptotically as good as the solution w_t^* of a batch algorithm trained on the same t examples.



$$\mathbf{E}\left[(w_t - w^*)^2\right] \sim \mathbf{E}\left[(w_t^* - w^*)^2\right] \sim \tfrac{K}{t}$$

Fig. 6. : After t iterations on fresh examples, the point w_t reached by an optimal stochastic learning algorithm is asymptotically as good as the solution w_t^* of a batch algorithm trained on the same t examples.

4.5 Comparing computational complexities

The discussion so far has established that a properly designed online learning algorithm performs as well as any batch learning algorithm for a same number of examples. We now establish that, given the same computing resources, a stochastic learning algorithm can asymptotically process more examples than a batch learning algorithm.

Each iteration of a batch learning algorithm running on N training examples requires a time $K_1 N + K_2$. Constants K_1 and K_2 respectively represent the time required to process each example, and the time required to update the parameters. Result (29) provides the following asymptotic equivalence:

$$\mathbf{E}\left[\,(w_N^* - w^*)^2\,\right] \sim \frac{1}{N}$$

The batch algorithm must perform enough iterations to approach the empirical optimum w_N^* with at least the same accuracy ($\sim 1/N$). A very efficient algorithm with quadratic convergence achieves this after a number of iterations asymptotically proportional to $(\log\log N)$.

Running a stochastic learning algorithm requires a constant time K_3 per processed example. Let us call T the number of examples processed by the stochastic learning algorithm using the same computing resources as the batch algorithm. We then have:

$$K_3 T \sim (K_1 N + K_2)\log\log N \quad \Longrightarrow \quad T \sim N\ \log\log N$$

The parameter w_T of the stochastic algorithm also converges according to (30). Comparing the accuracies of both algorithms shows that the stochastic algorithm asymptotically provides a better solution by a factor $\sim (\log\log N)$.

$$\mathbf{E}\left[\,(w_T - w^*)^2\,\right] \sim \frac{1}{N\ \log\log N} \ll \frac{1}{N} \sim \mathbf{E}\left[\,(w_N^* - w^*)^2\,\right] \qquad (31)$$

This $(\log\log N)$ factor corresponds to the number of iterations required by the batch algorithm. This number increases slowly with the desired accuracy of the solution. In practice, this factor is much less significant than the actual value of the constants K_1, K_2 and K_3. Experience shows however that stochastic algorithms are considerably easier to implement. Each iteration of the batch algorithm involves a large summation over all the available examples. Memory must be allocated to hold these examples. On the other hand, each iteration of the stochastic algorithm only involves one random example which can then be discarded.

4.6 Examples

Optimal Learning Rate for K-Means. Second derivative information can be used to determine very efficient learning rates for the K-Means algorithm (Section 2.4). A simple analysis of the loss function (16) shows that the Hessian of the cost function is a diagonal matrix [11] whose coefficients $\lambda(k)$ are equal to the probabilities that an example x is associated with the corresponding centroid $w(k)$.

These probabilities can be estimated by simply counting how many examples $n(k)$ have been associated with each centroid $w(k)$. Each iteration of the corresponding stochastic algorithm consists in drawing a random example x_t, finding the closest centroid $w(k)$, and updating both the count and the centroid with the following equations:

$$\left[\begin{array}{l} n_{t+1}(k) = n_t(k) + 1 \\ w_{t+1}(k) = w_t(k) + \frac{1}{n_{t+1}(k)}(x_t - w_t(k)) \end{array}\right. \tag{32}$$

Algorithm (32) very quickly locates the relative position of clusters in the data. Terminal convergence however is slowed down by the noise implied by the random choice of the examples. Experimental evidence [11] suggest that the best optimization speed is achieved by first using the stochastic algorithm (32) and then switching to a batch super-linear version of K-means.

Kalman Algorithms. The Kalman filter theory has introduced an efficient way to compute an approximation of the inverse of the Hessian of certain cost functions. This idea is easily demonstrated in the case of linear algorithms such as the Adaline (Section 2.3). Consider stochastic gradient descent applied to the minimization of the following mean square cost function:

$$C(w) \;=\; \int Q(z,w)\,dP(z) \qquad \text{with} \qquad Q(z,w) \;\stackrel{\triangle}{=}\; (y - w'x)^2 \tag{33}$$

Each iteration of this algorithm consists of drawing a new pair $z_t = (x_t, y_t)$ from the distribution $dP(z)$ and applying a parameter update formula similar to (27):

$$w_{t+1} \;=\; w_t - H_t^{-1}\,\nabla_w Q(z_t, w_t) \;=\; w_t - H_t^{-1}\,(y_t - w_t'x_t)'x_t \tag{34}$$

where H_t denotes the Hessian of an empirical estimate $C_t(w)$ of the cost function $C(w)$ based on the examples $z_1, \ldots, z_t$ observed so far.

$$C_t(w) \;\stackrel{\triangle}{=}\; \frac{1}{2}\sum_{i=1}^{t} Q(z_i, w) \;=\; \frac{1}{2}\sum_{i=1}^{t}(y_i - w'x_i)^2 \tag{35}$$

$$H_t \stackrel{\triangle}{=} \nabla_w^2 C_t(w) = \sum_{i=1}^{t} x_i x_i' \tag{36}$$

Directly computing the matrix H_t^{-1} at each iteration would be very expensive. We can take advantage however of the recursion $H_t = H_{t-1} + x_t x_t'$ using the well known matrix equality:

$$(A + BB')^{-1} = A^{-1} - (A^{-1}B)\,(I + B'A^{-1}B)^{-1}\,(A^{-1}B)' \tag{37}$$

Algorithm (34) then can be rewritten recursively using the Kalman matrix $K_t = H_{t-1}^{-1}$. The resulting algorithm (38) converges much faster than the delta rule (6) and yet remains quite easy to implement:

$$\left[\begin{array}{l} K_{t+1} = K_t - \dfrac{(K_t x_t)(K_t x_t)'}{1 + x_t' K_t x_t} \\ w_{t+1} = w_t - K_{t+1}\,(y_t - w_t' x_t)' x_t \end{array}\right. \tag{38}$$

Gauss Newton Algorithms. Non linear least mean square algorithms, such as the multi-layer networks (Section 2.3) can also benefit from non-scalar learning rates. The idea consists of using an approximation of the Hessian matrix. The second derivatives of the loss function (9) can be written as:

$$\begin{aligned} \frac{1}{2}\nabla_w^2\,(y - f(x,w))^2 &= \nabla_w f(x,w)\,\nabla_w' f(x,w) - (y - f(x,w))\nabla_w^2 f(x,w) \\ &\approx \nabla_w f(x,w)\,\nabla_w' f(x,w) \end{aligned} \tag{39}$$

Approximation (39), known as the *Gauss Newton Approximation*, neglects the impact of the non linear function f on the curvature of the cost function. With this approximation, the Hessian of the empirical stochastic cost takes a very simple form.

$$H_t(w) \approx \sum_{i=1}^{t} \nabla_w f(x_i, w)\,\nabla_w' f(x_i, w) \tag{40}$$

Although the real Hessian can be negative, this approximated Hessian is always positive, a useful property for convergence. Its expression (40) is reminiscent of the linear case (36). Its inverse can be computed using similar recursive equations.

Natural Gradient. Information geometry [1] provides an elegant description of the geometry of the cost function. It is best introduced by casting the learning problem as a density estimation problem. A multilayer perceptron $f(x,w)$, for instance, can be regarded as a parametric regression model $y = f(x,w) + \varepsilon$ where ε represents an additive Gaussian noise. The network function $f(x,w)$ then becomes part of the Gaussian location model:

$$p(z|w) = C_\sigma \exp\left(-\frac{(y - f(x,w))^2}{2\sigma^2} \right) \tag{41}$$

The optimal parameters are found by minimizing the Kullback-Leibler divergence between $p(z|w)$ and the ground truth $P(z)$. This is equivalent to the familiar optimization of the mean square loss (9):

$$\mathbf{E}_z \log \frac{P(z)}{p(z|w)} = \frac{1}{\sigma^2}\mathbf{E}_z Q_{mse}(z,w) + \text{Constant} \tag{42}$$

The essential idea consists of endowing the space of the parameters w with a distance that reflects the proximity of the distributions $p(z|w)$ instead of the proximity of the parameters w. Multilayer networks, for instance, can implement the same function with very different weights vectors. The new distance distorts the geometry of the parameter space in order to represent the closeness of these weight vectors.

The infinitesimal distance between distributions $p(z|w)$ and $p(z|w+dw)$ can be written as follows:

$$D(w||w+dw) \;\approx\; dw'\mathcal{G}(w)dw \tag{43}$$

where $\mathcal{G}(w)$ is the Fisher Information matrix:

$$\mathcal{G}(w) \;\stackrel{\triangle}{=}\; \int \left(\nabla_w \log p(z|w) \nabla_w \log p(z|w)'\right)\; p(z|w)dz$$

The determinant $|\mathcal{G}(w)|$ of the Fisher information matrix usually is a smooth function of the parameter w. The parameter space is therefore composed of Riemannian domains where $|\mathcal{G}(w)| \neq 0$ separated by critical sub-spaces where $|\mathcal{G}(w)| = 0$.

The *Natural Gradient* algorithm [3] provides a principled way to search a Riemannian domain. The gradient $\nabla_w C(w)$ defines the steepest descent direction in the Euclidean space. The steepest descent direction in a Riemannian domain differs from the Euclidexan one. It is defined as the vector dw which maximizes $C(w) - C(w+dw)$ in the δ-neighborhood:

$$D(w||w+dw) \;\approx\; dw'\mathcal{G}(w)dw \leq \delta. \tag{44}$$

A simple derivation then shows that multiplying the gradient by the inverse of the Fisher Information matrix yields the steepest Riemannian direction. The Natural Gradient algorithm applies the same correction to the stochastic gradient descent algorithm (4):

$$w_{t+1} \;=\; w_t - \gamma_t \mathcal{G}^{-1}(w_t) \nabla_w\, Q(z, w_t), \tag{45}$$

The similarity between the update rules (27) and (45) is obvious. This link becomes clearer when the Fisher Information matrix is written in Hessian form,

$$\mathcal{G}(w) \;\stackrel{\triangle}{=}\; \int - \left(\nabla^2_w \log p(z|w)\right)\; p(z|w)dz$$

where ∇^2_w denotes a second derivative. When the parameter approaches the optimum, distribution $p(z|w)$ becomes closer to the ground truth $dP(z)$, and the Fisher Information matrix $\mathcal{G}(w)$ aligns with the Hessian matrix $\nabla^2_w \mathbf{E}_z Q(z,w)$. The natural gradient asymptotically behaves like a second order algorithm.

Remark. The above algorithms are all derived from (27) and suffer from the same limitation. The number of coefficients in matrix Φ_t scales like the square of the number of parameters. Manipulating such large matrices often requires excessive computer time and memory.

Result (30) holds when $\Phi_t \longrightarrow H^{-1}(w^*)$. This implies that Φ_t must be a full rank approximation of H^{-1}. Suppose instead that Φ_t converges to a more economical approximation of H^{-1} involving a limited number of coefficients. With a proper choice of learning rates γ_t, such an approximate second order stochastic gradient algorithm keeps the $\mathcal{O}(1/t)$ behavior (30) with a worse constant $K_A > K$. Such a stochastic algorithm will eventually outperform batch algorithms because $(\log\log N)$ will eventually become larger than the ratio K_A/K. In practice this can take a very long time...

Approximate second order stochastic algorithms are still desirable because it might be simply impossible to simply store a full rank matrix Φ_t, and because manipulating the approximation of the Hessian might bring computational gains that compare well with ratio K_A/K. The simplest approximation [5] involves a diagonal approximation of Φ_t. More sophisticated schemes [32, 38] attempt to approximate the average value of $\Phi_t \nabla_w Q(z, w_t)$ using simpler calculations for each example.

5 Conclusion

A broad family of learning algorithms can be formalized as stochastic gradient descent algorithms. It includes numerous well known algorithms such as Perceptrons, Adalines, K-Means, LVQ, and Multi-Layer Networks as well as more ambitious learning systems such as Graph Transformer Networks.

All these algorithms share common convergence properties. In particular, stochastic gradient descent simultaneously is a very poor optimization algorithm and a very effective learning algorithm.

References

1. S.-I. Amari. *Differential-geometrical methods in statistics*. Springer Verlag, Berlin, New York, 1990.
2. S.I. Amari. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, EC-16:299–307, 1967.
3. Sun-Ichi Amari. Natural learning in structured parameter spaces – natural riemannian gradient. In *Neural Information Processing Systems*, volume 9, pages 127–133, Cambridge, MA., 1996. MIT Press.
4. Roberto Battiti. First- and second-order methods for learning: Between steepest descent and newton's method. *Neural Computation*, 4:141–166, 1992.
5. S. Becker and Y. Le Cun. Improving the convergence of back-propagation learning with second-order methods. In D. Touretzky, G. Hinton, and T Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 29–37, San Mateo, 1989. Morgan Kaufman.

6. Y. Bengio, Y. LeCun, C. Nohl, and C. Burges. Lerec: A nn/hmm hybrid for on-line handwriting recognition. *Neural Computation*, 7(6), November 1995.
7. A. Benveniste, M. Metivier, and P. Priouret. *Adaptive Algorithms and Stochastic Approximations*. Springer Verlag, Berlin, New York, 1990.
8. L. Bottou, Y. Le Cun, and Y. Bengio. Global training of document processing systems using graph transformer networks. In *Proc. of Computer Vision and Pattern Recognition*, pages 489–493, Puerto-Rico, 1997. IEEE.
9. Léon Bottou. *Une Approche théorique de l'Apprentissage Connexionniste: Applications à la Reconnaissance de la Parole*. PhD thesis, Université de Paris XI, Orsay, France, 1991.
10. Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.
11. Léon Bottou and Yoshua Bengio. Convergence properties of the kmeans algorithm. In *Advances in Neural Information Processing Systems*, volume 7, Denver, 1995. MIT Press.
12. Léon Bottou and Yann LeCun. Large scale online learning. In *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2004.
13. Léon Bottou and Yann LeCun. On-line learning for very large datasets. *Applied Stochastic Models in Business and Industry*, 2004. To appear, Special issue.
14. Léon Bottou and Noboru Murata. Stochastic approximations and efficient learning. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks, Second edition,*. The MIT Press, Cambridge, MA, 2002.
15. J. E., Jr. Dennis and R. B. Schnabel. *Numerical Methods For Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.
16. R.O. Duda and P.E. Hart. *Pattern Classification And Scene Analysis*. Wiley and Son, 1973.
17. C. Gentile and M. K. Warmuth. Linear hinge loss and average margin. In *Neural Information Processing Systems*, volume 11, pages 255–231, Cambridge, MA., 1999. MIT Press.
18. D. O. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.
19. T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.
20. T. Kohonen, G. Barna, and R. Chrisley. Statistical pattern recognition with neural network: Benchmarking studies. In *Proceedings of the IEEE Second International Conference on Neural Networks*, volume 1, pages 61–68, San Diego, 1988.
21. A. A. Krasovskii. *Dynamic of continuous self-Organizing Systems*. Fizmatgiz, Moscow, 1963. (in russian).
22. H. J. Kushner and D. S. Clark. *Stochastic Approximation for Constrained and Unconstrained Systems*. Applied Math. Sci. 26. Springer Verlag, Berlin, New York, 1978.
23. Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Winter 1989.
24. Yann Le Cun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient based learning applied to document recognition. *Proceedings of IEEE*, 86(11):2278–2324, 1998.
25. Yann Le Cun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science 1524. Springer Verlag, 1998.

26. Yann LeCun, Léon Bottou, and Jie HuangFu. Learning methods for generic object recognition with invariance to pose and lighting. In *Proc. of Computer Vision and Pattern Recognition*, Washington, D.C., 2004. IEEE.
27. L. Ljung and T. Söderström. *Theory and Practice of recursive identification*. MIT Press, Cambridge, MA, 1983.
28. J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. LeCam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics, and Probabilities*, volume 1, pages 281–297, Berkeley and Los Angeles, (Calif), 1967. University of California Press.
29. M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.
30. U. Müller, A. Gunzinger, and W. Guggenbühl. Fast neural net simulation with a DSP processor array. *IEEE Trans. on Neural Networks*, 6(1):203–213, 1995.
31. Noboru Murata and Sun-ichi Amari. Statistical analysis of learning dynamics. *Signal Processing*, 74(1):3–28, 1999.
32. Genevieve B. Orr and Todd K. Leen. Momentum and optimal stochastic search. In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman, and A. S. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*, pages 351–357. Lawrence Erlbaum Associates, 1994.
33. H. Robbins and S. Monro. A stochastic approximation model. *Ann. Math. Stat.*, 22:400–407, 1951.
34. F. Rosenblatt. The perceptron: A perceiving and recognizing automaton. Technical Report 85-460-1, Project PARA, Cornell Aeronautical Lab, 1957.
35. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition*, volume I, pages 318–362. Bradford Books, Cambridge, MA, 1986.
36. Ji Zhu Saharon Rosset and Trevor Hastie. Margin maximizing loss functions. In *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2004.
37. M. Schenkel, H. Weissman, I. Guyon, C. Nohl, and D. Henderson. Recognition-based segmentation of on-line hand-printed words. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 723–730, Denver, CO, 1993.
38. N. Schraudolph and T. Graepel. Conjugate directions for stochastic gradient descent. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN 2002).*, Berlin, 2002. Springer Verlag.
39. T. J. Sejnowski and C. R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.
40. Ya Tsypkin. *Adaptation and Learning in automatic systems*. Academic Press, New York, 1971.
41. Ya Tsypkin. *Foundations of the theory of learning systems*. Academic Press, New York, 1973.
42. V. N. Vapnik. *Estimation of dependences based on empirical data*. Springer Series in Statistics. Springer Verlag, Berlin, New York, 1982.
43. V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, Berlin, New York, 1995.
44. B. Widrow and M. E. Hoff. Adaptive switching circuits. In *IRE WESCON Conv. Record, Part 4.*, pages 96–104, 1960.
45. B. Widrow and S. D. Stearns. *Adaptive Signal Processing*. Prentice-Hall, 1985.
46. R. Wolf and J. Platt. Postal address block location using a convolutional locator network. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 745–752, 1994.