# Online Learning and Stochastic Approximations

*Léon Bottou*
AT&T Labs–Research
Red Bank, NJ07701

**Abstract**

The convergence of online learning algorithms is analyzed using the tools of the stochastic approximation theory, and proved under very weak conditions. A general framework for online learning algorithms is first presented. This framework encompasses the most common online learning algorithms in use today, as illustrated by several examples. The stochastic approximation theory then provides general results describing the convergence of all these learning algorithms at once.

## 1 Introduction

Almost all of the early work on *Learning Systems* focused on online algorithms (Hebb, 1949) (Rosenblatt, 1957) (Widrow and Hoff, 1960) (Amari, 1967) (Kohonen, 1982). 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 (LeCun et al., 1989) (Müller, Gunzinger and Guggenbühl, 1995).

The early *Recursive Adaptive Algorithms* were introduced during the same years (Robbins and Monro, 1951) and very often by the same people (Widrow and Stearns, 1985). First developed in the engineering world, recursive adaptation algorithms have turned into a mathematical discipline, namely *Stochastic Approximations* (Kushner and Clark, 1978) (Ljung and Söderström, 1983) (Benveniste, Metivier and Priouret, 1990).

Although both domains have enjoyed the spotlights of scientific fashion at different times and for different reasons, they essentially describe the same elementary ideas. Many authors of course have stressed this less-than-fortuitous similarity between learning algorithms and recursive adaptation algorithms (Mendel and Fu, 1970) (Tsypkin, 1971).

The present work builds upon this similarity. Online learning algorithms are analyzed using the stochastic approximation tools. Convergence is characterized under very weak conditions: the expected risk must be reasonably well behaved and the learning rates must decrease appropriately.

The main discussion describes a general framework for online learning algorithms, presents a number of examples, and analyzes their dynamical properties. Several comment sections illustrate how these ideas can be generalized and how they relate to other aspects of learning theory. In other words, the main discussion gives answers, while the comments raise questions. Casual readers may skip these comment sections.

## 2 A Framework for Online Learning Systems

The starting point of a mathematical study of online learning must be a mathematical statement for our subjective understanding of what a learning system is. It is difficult to agree on such a statement, because we are learning systems ourselves and often resent this mathematical reduction of an essential personal experience.

This contribution borrows the framework introduced by the Russian school (Tsypkin, 1971; Vapnik, 1982). This formulation can be used for understanding a significant number of online learning algorithms, as demonstrated by the examples presented in section 3.

### 2.1 Expected Risk Function

In (Tsypkin, 1971; Tsypkin, 1973), the goal of a learning system consists of finding the minimum of a function $J(w)$ named the *expected risk function*. This function is decomposed as follows:

$$J(w) \;\stackrel{\triangle}{=}\; \mathbf{E}_z\, Q(z,w) \;\stackrel{\triangle}{=}\; \int Q(z,w)\; dP(z) \tag{2.1}$$

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 $\mathcal{Z}$.

The occurrence of the events $z$ is modeled as random independent observations drawn from an unknown probability distribution $dP(z)$ named the *grand truth distribution*. The risk function $J(w)$ is simply the expectation of the loss function $Q(z,w)$ for a fixed value of the parameter $w$. This risk function $J(w)$ is poorly defined because the grand truth distribution $dP(z)$ is unknown by hypothesis.

Consider for instance a neural network system for optical ZIP code recognition, as described in (LeCun et al., 1989). An observation $z$ is a pair $(x,y)$ composed of a ZIP code image $x$ and its intended interpretation $y$. Parameters $w$ are the adaptable weights of the neural network. The loss function

$Q(z, w)$ measures the economical cost (in hard currency units) of delivering a letter marked with ZIP code $z$ given the answer produced by the network on image $x$. This cost is minimal when the network gives the right answer. Otherwise the loss function measures the higher cost of detecting the error and re-routing the letter.

### Comments

Probabilities are used in this framework for representing the unknown truth underlying the occurrences of observable events. Using successive observations $z_t$, the learning system will uncover a part of this truth in the form of parameter values $w_t$ that hopefully decrease the risk functional $J(w_t)$. This use of probabilities is very different from the Bayesian practice, where a probability distribution represents the increasing knowledge of the learning system. Both approaches however can be re-conciliated by defining the parameter space $\mathcal{W}$ as a another space of probability distributions. The analysis then must carefully handle two different probability distributions with very different meanings.

In this framework, every known fact about the real world should be removed from distribution $dP(z)$ by properly redefining the observation space $\mathcal{Z}$ and of the loss function $Q(z, w)$. Consider for instance that a known fraction of the ZIP code images are spoiled by the image capture system. An observation $z$ can be factored as a triple $(\kappa, x, y)$ composed of an envelope $x$, its intended ZIP code $y$, and a binary variable $\kappa$ indicating whether the ZIP code image is spoiled. The loss function can be redefined as follows:

$$\begin{aligned} J(w) &= \int Q(z,w)\ dP(\kappa, x, y) \\ &= \int \left( \int Q(z,w)\, dP(\kappa | x, y) \right)\ dP(x, y) \end{aligned}$$

The inner integral in this decomposition is a new loss function $Q'(x, y, w)$ which measures the system performance on redefined observations $(x, y)$. This new loss function accounts for the known deficiencies of the image capture system. This factorization technique reveals a new probability distribution $dP(x, y)$ which is no longer representative of this a priori knowledge.

This technique does not apply to knowledge involving the learning system itself. When we say for instance that an unknown function is smooth, we mean that it pays to bias the learning algorithm towards finding smoother functions. This statement does not describe a property of the grand truth distribution. Its meaning is attached to a particular learning system. It does not suggests a redefinition of the problem. It merely suggests a modification of the learning system, like the introduction of a regularization parameter.

## 2.2 Gradient Based Learning

The expected risk function (2.1) cannot be minimized directly because the grand truth distribution is unknown. It is however possible to compute an

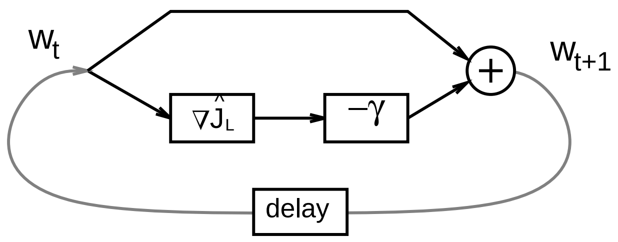

**Figure 1**: Batch Gradient Descent. The parameters of the learning system are updated using the gradient of the empirical risk $\hat{J}_L$ defined on the training set.

approximation of $J(w)$ by simply using a finite *training set* of independent observations $z_1, \ldots, z_L$.

$$J(w) \;\approx\; \hat{J}_L(w) \;\stackrel{\triangle}{=}\; \frac{1}{L}\sum_{n=1}^{L} Q(z_n, w) \tag{2.2}$$

General theorems (Vapnik, 1982) show that minimizing the *empirical risk* $\hat{J}_L(w)$ can provide a good estimate of the minimum of the expected risk $J(w)$ 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.

### 2.2.1 Batch Gradient Descent

Minimizing the empirical risk $\hat{J}_L(w)$ can be achieved using a *batch gradient descent* algorithm. Successive estimates $w_t$ of the optimal parameter are computed using the following formula (figure 1) where the learning rate $\gamma_t$ is a positive number.

$$w_{t+1} \;=\; w_t - \gamma_t \nabla_w \hat{J}_L(w_t) \;=\; w_t - \gamma_t \frac{1}{L}\sum_{i=1}^{L} \nabla_w Q(z_n, w_t) \tag{2.3}$$

The properties of this optimization algorithm are well known (section 4.2). When the learning rate $\gamma_t$ are small enough, the algorithm converges towards a local minimum of the empirical risk $\hat{J}_L(w)$. Considerable convergence speedups can be achieved by replacing the learning rate $\gamma_t$ by a suitable definite positive matrix (Dennis and Schnabel, 1983).

Each iteration of the batch gradient descent algorithm (figure 1) 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.

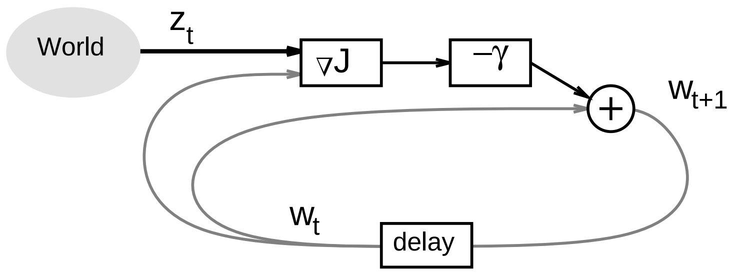

**Figure 2**: Online Gradient Descent. The parameters of the learning system are updated using information extracted from real world observations.

### 2.2.2 Online Gradient Descent

The elementary *online gradient descent* algorithm is obtained by dropping the averaging operation in the batch gradient descent algorithm (2.3). 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.

$$w_{t+1} \ = \ w_t - \gamma_t \nabla_w \, Q(z_t, w_t) \tag{2.4}$$

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.

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 2. This formulation 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. An airplane autopilot, for instance, may continuously learn how commands affect the route of the airplane. Such a system would compensate for changes in the weather or in petrol weight.

### Comments

Formulating online gradient descent without reference to a training set 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 direct optimization shortcuts the usual discussion about differences between optimizing the empirical risk and the expected risk (Vapnik, 1982; Vapnik, 1995). 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. Discussing the convergence speed of such an online algorithm provides an alternative to the Vapnik-Chervonenkis bounds.

This alternative comes with severe restrictions. The convergence proofs proposed here (section 5) only address the convergence of the algorithm towards a local minimum. We can safely conjecture that a general study of the convergence of an online algorithm towards a global minimum should handle the central concepts of the necessary and sufficient conditions for the consistency of a learning algorithm (Vapnik, 1995).

## 2.3 General Online Gradient Algorithm

The rest of this contribution addresses a single *general online gradient algorithm* algorithm for minimizing the following cost function $C(w)$.

$$C(w) \;\stackrel{\triangle}{=}\; \mathbf{E}_z Q(z,w) \;\stackrel{\triangle}{=}\; \int Q(z,w)\,dP(z) \tag{2.5}$$

Each iteration of this algorithm consists of drawing an event $\mathbf{z}_t$ from distribution $dP(z)$ and applying the following update formula.

$$w_{t+1} \;=\; w_t - \gamma_t H(\mathbf{z}_t, w_t) \tag{2.6}$$

The learning rates $\gamma_t$ are either positive numbers or definite positive matrices. The update term $H(\mathbf{z}, w)$ fulfills the following condition.

$$\mathbf{E}_{\mathbf{z}}\, H(\mathbf{z}, w) \;=\; \nabla_w\, C(w) \tag{2.7}$$

The distribution function $dP(z)$ can be understood as the grand truth distribution. The cost function $C(w)$ minimized by this algorithm is then equal to the expected risk $J(w)$. This setup addresses the adaptive version of the online gradient descent, without reference to a training set.

All results however remain valid if we consider a discrete distribution function defined on a particular training set $\{z_1, \ldots, z_L\}$. The cost function $C(w)$ minimized by this algorithm is then equal to the empirical risk $\hat{J}_L$. This second setup addresses the use of online gradient descent for optimizing the training error defined on a finite training set.

### Comments

Typography conscious readers will notice the subtle difference between the observable events $z$ used in the cost function (2.5) and the events $\mathbf{z}$ drawn before each iteration of the algorithm (2.6). In the simplest case indeed, these two variables represent similar objects: a single example is drawn before each iteration of the online gradient descent algorithm. The framework described above also applies to more complex cases like *mini-batch* or *noisy* gradient descent. Mini-batch gradient descent uses several examples for each iteration, collectively referred to as $\mathbf{z}_t$. Noisy gradient descent uses a noisy update term $\nabla_w C(w_t) + \xi_t$. The analysis presented in this contribution holds as long as the update term fulfills condition (2.7).

Finally the examples $\mathbf{z}_t$ are assumed to be independently drawn from a single probability distribution function $dP(z)$. In practice however, examples are often chosen sequentially in a training set. There are tools indeed for dealing with examples $\mathbf{z}_t$ drawn using a Markovian process (Benveniste, Metivier and Priouret, 1990).

## 3 Examples

This section presents a number of examples illustrating the diversity of learning algorithms that can be expressed as particular cases of the general online gradient descent algorithm (section 2.3). More classical algorithms can be found in (Tsypkin, 1971).

Some algorithms were designed with a well defined cost function, like the adaline (section 3.1.1) or the multi-layer perceptron (section 3.1.2). Other algorithms did not initially refer to a particular cost function, but can be reformulated as online gradient descent procedures, like $K$-Means (section 3.2.2) or LVQ2 (section 3.2.3). The cost function then provides a useful characterization of the algorithm. Finally, certain algorithms, like Kohonen's topological maps (Kohonen, 1982), are poorly represented as the minimization of a cost function. Yet some authors have found useful to coerce these algorithms into an online gradient descent anyway (Schumann and Retzko, 1995).

### 3.1 Online Least Mean Squares

#### 3.1.1 Widrow's Adaline

The *adaline* (Widrow and Hoff, 1960) 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 3) learning algorithm adapts the parameters of a single *threshold element*. 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 $\beta$ then is represented as an extra coefficient in the parameter vector $w$. With

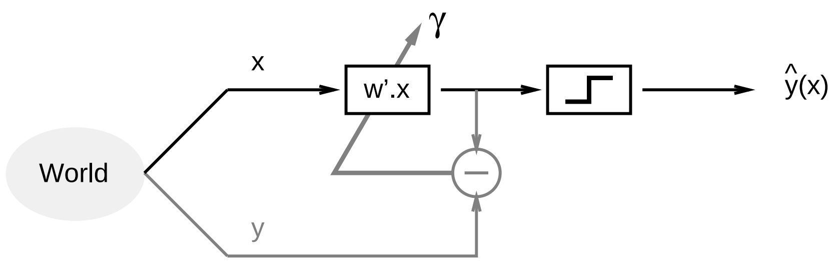

**Figure 3**: Widrow's Adaline. The adaline computes a binary indicator by thresholding a linear combination of its input. Learning is achieved using the *delta rule*.

this convention, the output of the threshold element can be written as

$$\hat{y}_w(x) \;\stackrel{\triangle}{=}\; \mathrm{sign}(w'x) \;=\; \mathrm{sign}\sum_i w_i x_i \tag{3.1}$$

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):

$$w_{t+1} = w_t - \gamma_t(y_t - w_t'x_t)'x_t \tag{3.2}$$

This delta rule is nothing more than an iteration of the online gradient descent algorithm (2.4) with the following loss function:

$$Q_{\mathrm{adaline}}(z, w) \;\stackrel{\triangle}{=}\; (y - w'x)^2 \tag{3.3}$$

This loss function does not take the discontinuity of the threshold element (3.1) 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 element.

### 3.1.2 Multi-Layer Networks

Multi-layer networks were initially designed to overcome the computational limitation of the threshold elements (Minsky and Papert, 1969). Arbitrary binary mappings can be implemented by stacking several layers of threshold elements, each layer using the outputs of the previous layers elements 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 (Rumelhart, Hinton and Williams, 1986) consisted of noticing that the discontinuity of the threshold element could be represented by a smooth non-linear approximation.

$$\text{sign}(w'x) \;\approx\; \tanh(w'x) \tag{3.4}$$

Using such *sigmoidal elements* 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 elements 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, (Rumelhart, Hinton and Williams, 1986) provides an efficient way to compute the gradients of the mean square loss function.

$$Q_{\text{mse}}(z,w) = \frac{1}{2}\left(y - f(x,w)\right)^2 \tag{3.5}$$

Both the batch gradient descent (2.3) and the online gradient descent (2.4) 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 (Müller, Gunzinger and Guggenbühl, 1995). 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 (2.3) averages the gradient of the loss function over the whole training set, causing redundant computations. On the other hand, running online gradient descent (2.4) on all examples of the training set would amount to performing two complete learning iterations over the duplicated subset.

## 3.2 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 intuition can be formalized using the general online gradient descent algorithm (2.6). The general algorithm can use any update term $H(\mathbf{z},w)$ which fulfills condition (2.7). We assume here that the cost function $C(w)$ is made differentiable when the loss function $Q(z,w)$ is integrated with the probability distribution $dP(z)$.

The following update term amounts to drawing another example whenever

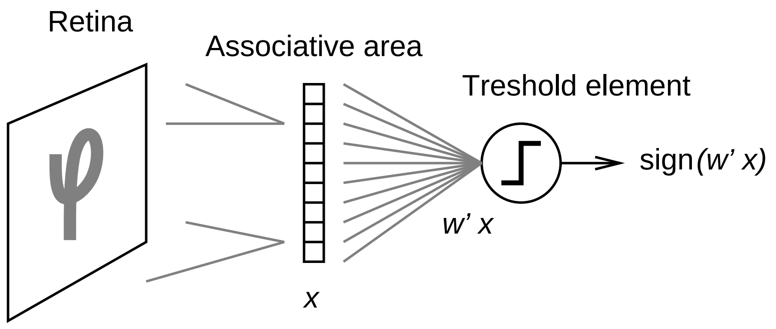

**Figure 4**: Rosenblatt's Perceptron is composed of a fixed preprocessing and of a trainable threshold element.

we reach a non differentiable point of the loss function.

$$H(z,w) = \begin{cases} \nabla_w Q(z,w) & \text{when differentiable} \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

For each parameter value $w$ reached by the algorithm, we assume that the loss function $Q(z,w)$ is differentiable everywhere except on a subset of examples $z$ with probability zero. Condition (2.7) then can be rewritten using (3.6) and explicit integration operators.

$$\int H(z,w)\, dP(z) = \int \nabla_w Q(z,w)\, dP(z) \;\overset{?}{=}\; \nabla_w \int Q(z,w)\, dP(z) \tag{3.7}$$

The Lebesgue integration theory provides a sufficient condition for swapping the integration ($\int$) and differentiation ($\nabla_w$) operators. 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:

$$\forall z,\ \forall\, v \in \vartheta(w),\quad |Q(z,v) - Q(z,w)| \;\leq\; |w - v|\, \Phi(z,w) \tag{3.8}$$

This condition (3.8) 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 (3.8) is a sufficient condition for safely ignoring a few non differentiable points.

### 3.2.1 Rosenblatt's Perceptron

During the early days of the computer age, the *perceptron* (Rosenblatt, 1957) generated considerable interest as a possible architecture for general purpose computers. This interest faded after the disclosure of its computational limitations (Minsky and Papert, 1969). Figure 4 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 element* (section 3.1.1).

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

$$w_{t+1} = w_t + 2\gamma_t y_t \, w_t' x_t \tag{3.9}$$

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

$$Q_{\text{perceptron}}(z, w) = (\text{sign}(w'x) - y) \, w'x \tag{3.10}$$

Although this loss function is non differentiable when $w'x$ is null, is meets condition (3.8) as soon as the expectation $\mathbf{E}(x)$ is defined. We can therefore ignore the non differentiability and apply the online gradient descent algorithm:

$$w_{t+1} = w_t - \gamma_t(\text{sign}(w_t'x_t) - y_t) \, x_t \tag{3.11}$$

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 (3.11) is equivalent to the perceptron rule (3.9).

The perceptron loss function (3.10) 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. If the training set is linearly separable (i.e. a threshold element can achieve zero misclassification) the perceptron algorithm finds a linear separation with probability one. Otherwise, the weights $w_t$ quickly tend towards zero.

### 3.2.2 $K$-Means

The $K$-Means algorithm (MacQueen, 1967) 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.

$$Q_{\text{kmeans}}(x, w) \;\stackrel{\triangle}{=}\; \min_{k=1}^{K} \, (x - w_{(k)})^2 \tag{3.12}$$

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.

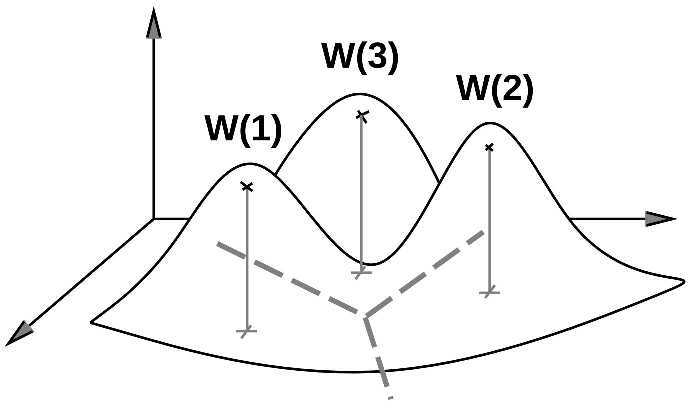

**Figure 5**: $K$-Means dispatches a predefined number of cluster centroids in a way that minimizes the quantification error.

This loss function is not differentiable on points located on the Voronoï boundaries of the set of centroids, but meets condition (3.8) 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.

$$w^-_{t+1} = w^-_t + \gamma_t(x_t - w^-_t) \tag{3.13}$$

This formula describes an elementary iteration of the $K$-Means algorithm. A very efficient choice of learning rates $\gamma_t$ will be suggested in section 3.3.2.

### 3.2.3 Learning Vector Quantization II

Kohonen's LVQ2 rule (Kohonen, Barna and Chrisley, 1988) 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. 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:

$$\begin{aligned} &\text{if } \begin{cases} x \text{ is misclassified } (w^- \neq w^+) \\ \text{and } \ (x - w^+)^2 < (1+\delta)(x - w^-)^2 \end{cases} \\ &\qquad \text{then } \begin{cases} w^-_{t+1} = w^-_t - \varepsilon_t(x - w^-_t) \\ w^+_{t+1} = w^+_t + \varepsilon_t(x - w^+_t) \end{cases} \end{aligned} \tag{3.14}$$

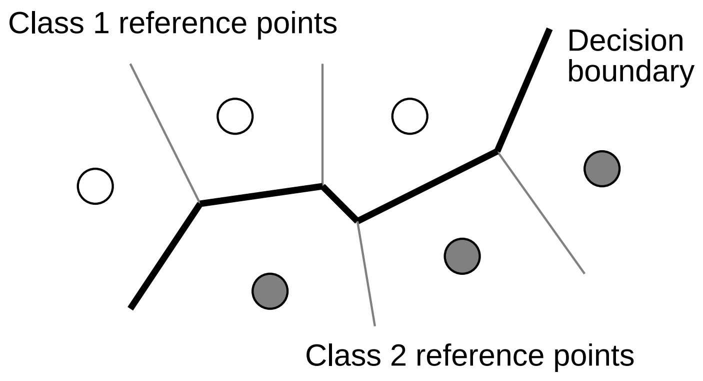

**Figure 6**: Kohonen's LVQ2 pattern recognition scheme outputs the class associated with the closest reference point to the input pattern.

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 $\delta$. 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:

$$Q_{\mathrm{lvq2}}(z,w) \triangleq \begin{cases} 0 & \text{if } x \text{ is well classified } (w^+ = w^-) \\ 1 & \text{if } (x-w^+)^2 \geq (1+\delta)(x-w^-)^2 \\ \frac{(x-w^+)^2-(x-w^-)^2}{\delta(x-w^-)^2} & \text{otherwise} \end{cases} \tag{3.15}$$

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 (Bottou, 1991). 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 (3.8) 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:

$$\begin{array}{l} \text{if } \begin{cases} x \text{ is misclassified } (w^- \neq w^+) \\ \text{and } \ (x-w^+)^2 < (1+\delta)(x-w^-)^2 \end{cases} \\ \qquad\qquad \text{then } \begin{cases} w^-_{t+1} = w^-_t - \gamma_t k_1 (x - w^-_t) \\ w^+_{t+1} = w^+_t + \gamma_t k_2 (x - w^+_t) \end{cases} \end{array} \tag{3.16}$$

$$\text{with } \ k_1 = \frac{1}{\delta(X-w^-)^2} \ \text{ and } \ k_2 = k_1 \frac{(X-w^+)^2}{(X-w^-)^2} \tag{3.17}$$

This online gradient descent algorithm (3.16) is equivalent to the usual LVQ2 learning algorithm (3.14). The two scalar coefficients $k_1$ and $k_2$ merely modify the proper schedule for the decreasing learning rates $\gamma_t$.

## 3.3 Quasi-Newton Online Algorithms

Both theoretical and empirical evidences demonstrate that batch gradient descent algorithms converge much faster when the scalar learning rates $\gamma_t$ are replaced by definite positive symmetric matrices that approximate the inverse of the Hessian of the cost function. The so-called *super-linear* algorithms achieve very high terminal convergence speed: the number of correct figures in the numerical representation of the solution increases exponentially (Dennis and Schnabel, 1983).

The same techniques are also effective for speeding up online gradient algorithms. The results however are much less impressive than those achieved with batch algorithms. No online gradient descent algorithm can achieve super-linear convergence (cf. comments to section 4). The terminal convergence of an online gradient algorithm is limited by the size of the learning rates. As will be shown in sections 4 and 5, decreasing the learning rates too quickly can prevent convergence.

The accuracy of a super-linear algorithm however is largely irrelevant to a learning system. Severe approximations, such as using a finite training set, would spoil the benefits of such an algorithm. Practitioners prefer techniques blessed with robust convergence properties, such as the Levenberg-Marquardt algorithm (Dennis and Schnabel, 1983). Furthermore, storing and processing a full learning rate matrix quickly becomes expensive when the dimension of the parameter vector $w$ increases. Less efficient approximations have been designed (Becker and LeCun, 1989) and have proven effective enough for large size applications (LeCun et al., 1989).

### 3.3.1 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 3.1.1). Consider online gradient descent applied to the minimization of the following mean square cost function:

$$C(w) \;=\; \int Q(z,w)\,dP(z) \quad \text{with} \quad Q(z,w) \;\triangleq\; (y - w'x)^2 \tag{3.18}$$

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

$$w_{t+1} \;=\; w_t - H_t^{-1}\,\nabla_w Q(z_t, w_t) \;=\; w_t - H_t^{-1}\,(y_t - w_t'x_t)'x_t \tag{3.19}$$

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

$$C_t(w) \triangleq \frac{1}{2}\sum_{i=1}^{t} Q(z_i, w) = \frac{1}{2}\sum_{i=1}^{t} (y_i - w'x_i)^2 \tag{3.20}$$

$$H_t \triangleq \nabla_w^2 C_t(w) = \sum_{i=1}^{t} x_i x_i' \tag{3.21}$$

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:

$$(A + BB')^{-1} = A^{-1} - (A^{-1}B)\,(I + B'A^{-1}B)^{-1}\,(A^{-1}B)' \tag{3.22}$$

Algorithm (3.19) then can be rewritten recursively using the *Kalman matrix* $K_t = H_{t-1}^{-1}$. The resulting algorithm (3.23) converges much faster than the delta rule (3.2) and yet remains quite easy to implement:

$$\left[\begin{array}{lcl} K_{t+1} & = & K_t - \dfrac{(K_t x_t)(K_t x_t)'}{1 + x_t' K_t x_t} \\ w_{t+1} & = & w_t - K_{t+1}\,(y_t - w_t' x_t)' x_t \end{array}\right. \tag{3.23}$$

## Comments

This linear algorithm has an interesting optimality property (Tsypkin, 1973). Because the cost function (3.20) is exactly quadratic, it is easy to prove by induction that (3.23) minimizes the online empirical cost $C_t(w)$ at each iteration. Assuming that $w_t$ is the minimum of $C_{t-1}(w)$, the following derivation shows that $w_{t+1}$ is the minimum of $C_t(w)$.

$$\begin{aligned} \nabla_w C_t(w_{t+1}) &= \nabla_w C_t(w_t) - H_t\,(w_{t+1} - w_t) \\ &= \nabla_w C_{t-1}(w_t) + \nabla_w Q(z_t, w_t) - H_t\,H_t^{-1} \nabla_w Q(z_t, w_t) \\ &= 0 \end{aligned}$$

Although this property illustrates the rapid convergence of algorithm (3.23), it only describes how the algorithm tracks an empirical approximation (3.20) of the cost function. This approximation may not provide very good generalization properties (Vapnik, 1995).

Non linear least mean square algorithms, such as the multi-layer networks (section 3.1.2) 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 (3.5) can be written as:

$$\begin{aligned} \frac{1}{2}\nabla_w^2\,(y - f(x,w))^2 &= \nabla_w f(x,w)\,\nabla_w' f(x,w) - (y - f(x,w))\nabla_w^2 f(x,w) \\ &\approx \nabla_w f(x,w)\,\nabla_w' f(x,w) \end{aligned} \tag{3.24}$$

Approximation (3.24), 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 online cost takes a very simple form.

$$H_t(w) \;\approx\; \sum_{i=1}^{t} \nabla_w f(x_i, w)\, \nabla'_w f(x_i, w) \tag{3.25}$$

Although the real Hessian can be negative, this approximated Hessian is always positive, a useful property for convergence. Its expression (3.25) is reminiscent of the linear case (3.21). Its inverse can be computed using similar recursive equations.

### 3.3.2 Optimal Learning Rate for $K$-Means

Second derivative information can also be used to determine very efficient learning rates for the $K$-Means algorithm (section 3.2.2). A simple analysis of the loss function (3.12) shows that the Hessian of the cost function is a diagonal matrix (Bottou and Bengio, 1995) 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 online 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:

$$\left[\begin{array}{lcl} n_{t+1}(k) &=& n_t(k) + 1 \\ w_{t+1}(k) &=& w_t(k) + \frac{1}{n_{t+1}(k)}\big(x_t - w_t(k)\big) \end{array}\right. \tag{3.26}$$

Algorithm (3.26) 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 (Bottou and Bengio, 1995) suggest that the best convergence speed is obtained by first using the online algorithm (3.26) and then switching to a batch super-linear version of $K$-means.

## 4 Convex Online Optimization

The next two sections illustrate how nicely the convergence of online learning algorithm is analyzed by the modern mathematical tools designed for stochastic approximations. This particular section addresses a simple convex case, while focusing on the mathematical tools and on their relation with the classical analysis of batch algorithms. This presentation owes much to a remarkable lecture by Michel Metivier (Metivier, 1981).

## 4.1 General Convexity

The analysis presented in this section addresses the convergence of the general online gradient algorithm (section 2.3) applied to the optimization of a differentiable cost function $C(w)$ with the following properties:

- The cost function $C(w)$ has a single minimum $w^*$.
- The cost function $C(w)$ satisfies the following condition:

$$\forall \varepsilon > 0, \quad \inf_{(w-w^*)^2>\varepsilon} (w - w^*)\, \nabla_w C(w) > 0 \tag{4.1}$$

Condition (4.1) simply states that the opposite of the gradient $-\nabla_w C(w)$ always points towards the minimum $w^*$. This particular formulation also rejects cost functions which have plateaus on which the gradient vanishes without making us closer to the minimum.

This condition is weaker than the usual notion of convexity. It is indeed easy to think of a non convex cost function which has a single minimum and satisfies condition (4.1). On the other hand, proving that all differentiable strictly convex functions satisfy this condition is neither obvious nor useful.

## 4.2 Batch Convergence Revisited

The convergence proof for the general online learning algorithm follow exactly the same three steps than the convergence proofs for batch learning algorithms. These steps consist of (a) defining a *Lyapunov* criterion of convergence, (b) proving that this criterion converges, and (c) proving that this convergence implies the convergence of the algorithm. These steps are now illustrated in the cases of the continuous gradient descent and the batch gradient descent.

### 4.2.1 Continuous Gradient Descent

The *continuous gradient descent* is a mathematical description of the ideal convergence of a gradient descent algorithm. A differential equation defines the parameter trajectory $w(t)$ as a continuous function of the time.

$$\frac{\mathrm{d}w}{\mathrm{d}t} \;=\; -\nabla_w C(w) \tag{4.2}$$

**Step a.** The convergence proof begins with the definition of a *Lyapunov function*, i.e. a positive function which indicates how far we are from the target.

$$h(t) \;\stackrel{\triangle}{=}\; (w(t) - w^*)^2 \tag{4.3}$$

**Step b.** Computing the derivative of $h(t)$ shows that the Lyapunov function $h(t)$ is a monotonically decreasing function.

$$\frac{\mathrm{d}h}{\mathrm{d}t} = 2(w - w^*)\frac{\mathrm{d}w}{\mathrm{d}t} = -2(w - w^*)\nabla_w C(w) \leq 0 \tag{4.4}$$

Since $h(t)$ is a positive decreasing function, it has a limit when $t \to \infty$.

**Step c.** Since the monotonic function $h(t)$ converges when $t$ grows, its gradient tends towards zero.

$$\frac{\mathrm{d}h}{\mathrm{d}t} = -2(w - w^*)\nabla_w C(w) \underset{t\to\infty}{\longrightarrow} 0 \tag{4.5}$$

Let us assume that the Lyapunov function $h(t)$ converges to a value greater than zero. After a certain time, the distance $h(t) = (w(t) - w^*)^2$ would remain greater than some positive value $\varepsilon$. This result is incompatible with condition (4.1) and result (4.5). The Lyapunov function $h(t)$ therefore converges to zero. This result proves the convergence of the continuous gradient descent (4.2).

$$w(t) \underset{t\to\infty}{\longrightarrow} w^* \tag{4.6}$$

### 4.2.2 Discrete Gradient Descent

The batch gradient descent algorithm has been introduced in section 2.2.1 in the context of learning algorithms. The cost function $C(w)$ is minimized by iteratively applying the following parameter update:

$$w_{t+1} \;=\; w_t - \gamma_t \nabla_w C(w) \tag{4.7}$$

Equation (4.7) is a discrete version of the continuous gradient descent (4.2). Although the discrete dynamics brings new convergence issues, the analysis of the convergence follows the same three elementary steps.

**Step a.** The convergence proof begins with the definition of a *Lyapunov sequence*, i.e. a sequence of positive numbers whose value measure how far we are from our target.

$$h_t \;\triangleq\; (w_t - w^*)^2 \tag{4.8}$$

**Lemma.** It is useful at this point to introduce a sufficient criterion for the convergence of a positive sequence $(u_t)$. Intuitively, a sequence $(u_t)$ converges when it is bounded and when its oscillations are damped. The oscillations can be monitored by summing the variations $u_t - u_{t-1}$ whenever $u_t > u_{t-1}$. These positive variations are represented with thick lines in figure 7. When the infinite sum of the positive variations converges, we are certain that the oscillations are damped. If all terms of the sequence are positive, this condition also ensures that the sequence if bounded.

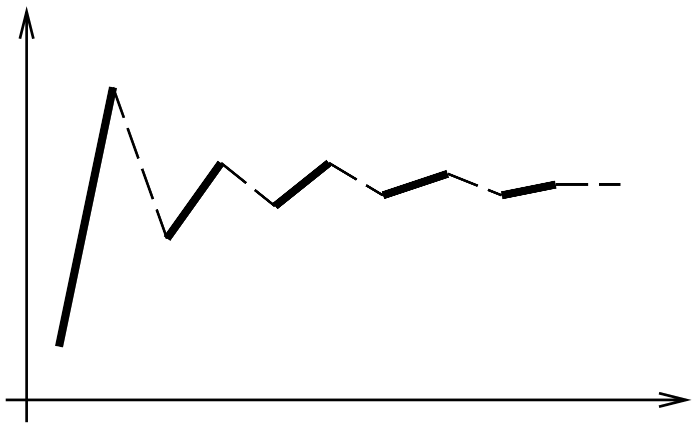

**Figure 7**: The convergence of the infinite sum of the positive increases (thick lines) is a sufficient (although not necessary) condition for the convergence of a positive sequence $h_t$. This condition ensures *(i)* that the sequence is bounded, and *(ii)* that the oscillations are damped.

This intuition is easily formalized by decomposing a term $u_t$ of a sequence using the sum $S_t^+$ of the *positive variations*:

$$S_t^+ \triangleq \sum_{i=1}^{t-1} (u_{i+1} - u_i)_+ \quad \text{with} \quad (x)_+ \triangleq \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

and the sum $S_t^-$ of the *negative variations*:

$$S_t^- \triangleq \sum_{i=1}^{t-1} (u_{i+1} - u_i)_- \quad \text{with} \quad (x)_- \triangleq \begin{cases} 0 & \text{if } x > 0 \\ x & \text{otherwise} \end{cases} \tag{4.10}$$

If the sum of the positive variations converges to $S_\infty^+$, this decomposition provides an upper bound for the positive sequence $u_t$.

$$0 \leq u_t = u_1 + S_t^+ + S_t^- \leq u_1 + S_\infty^+ + S_t^- < u_0 + S_\infty^+ \tag{4.11}$$

Furthermore, since $u_t \geq 0$, the same decompositions also provides a lower bound for the sum of the negative variations $S_t^-$.

$$0 - u_1 - S_\infty^+ \leq S_t^- \leq 0 \tag{4.12}$$

Since $S_t^-$ is a bounded monotonically decreasing sequence, it converges to a limit $S_\infty^-$. Since both sequences $S_t^+$ and $S_t^-$ converge, the sequence $u_t$ converges to $u_\infty = u_1 + S_\infty^+ + S_\infty^-$.

$$\left.\begin{array}{c} \forall t,\ u_t \geq 0 \\ \sum_{t=1}^{\infty} (u_{t+1} - u_t)_+ < \infty \end{array}\right\} \Longrightarrow u_t \underset{t\to\infty}{\longrightarrow} u_\infty \geq 0 \tag{4.13}$$

The convergence of the infinite sum of the positive variations is therefore a sufficient condition for the convergence of the sequence. Since the positive variations are positive, it is sufficient to prove that they are bounded by the summand of a convergent infinite sum.

**Step b.** The second step consists in proving that the Lyapunov sequence $(h_t)$ converges. Using the the definition (4.8) and from the gradient descent algorithm (4.7), we can write an expression for the variations of the Lyapunov criterion.

$$h_{t+1} - h_t \;=\; -2\gamma_t\,(w_t - w^*)\nabla_w C(w_t) \;+\; \gamma_t^2\,(\nabla_w C(w_t))^2 \tag{4.14}$$

The convexity criterion (4.1) ensures that the first term of this expression is always negative. Unlike the continuous variations (4.4), this expression contains a positive second term which reflects the discrete dynamics of the algorithm.

Additional conditions must be introduced in order to contain the effects of this second term. The first condition (4.15) states that the learning rates $\gamma_t$ decrease fast enough. This is expressed by the convergence of the infinite sum of the squared learning rates.

$$\sum_{i=1}^{\infty} \gamma_t^2 < \infty \tag{4.15}$$

The second condition (4.16) ensures that the size of gradients do not grow too fast when we move away from the minimum. This linear condition is met as soon as the eigenvalues of the Hessian matrix are bounded.

$$(\nabla_w C(w))^2 \leq A + B(w - w^*)^2 \quad A, B \geq 0 \tag{4.16}$$

Such a condition is required because the polynomial decrease of the learning rates would be too easily canceled by exponentially growing gradients. We can now transform equation (4.14) using the bound on the size of the gradients (4.16).

$$h_{t+1} - (1 - \gamma_t^2 B)h_t \leq -2\gamma_t\,(w_t - w^*)\nabla_w C(w_t) \;+\; \gamma_t^2 A \leq \gamma_t^2 A \tag{4.17}$$

We now define two auxiliary sequences $\mu_t$ and $h'_t$:

$$\mu_t \;\triangleq\; \prod_{i=1}^{t} \frac{1}{1 - \gamma_i^2 B} \;\xrightarrow[t\to\infty]{}\; \mu_\infty \quad \text{and} \quad h'_t \;\triangleq\; \mu_t h_t \tag{4.18}$$

The convergence of $\mu_t$ is easily verified by writing $\log \mu_t$ and using condition (4.15). Multiplying both the left-hand-side and the right hand side of (4.17) by $\mu_t$, we obtain:

$$(h'_{t+1} - h'_t) \leq \gamma_t^2 \mu_t A \tag{4.19}$$

Since the right hand side of (4.19) is positive, the positive variations of $h'_t$ are at most equal to $\gamma_t^2 \mu_t A$, which is the summand of a convergent infinite sum. According to lemma (4.13), the sequence $h'_t$ converges. Since $\mu_t$ converges, this convergence implies the convergence of the Lyapunov sequence $h_t$.

**Step c.** We now prove that the convergence of the Lyapunov sequence implies the convergence of the discrete gradient descent algorithm. Since the sequence $h_t$ converges, equation (4.17) implies the convergence of the following sum:

$$\sum_{i=1}^{\infty} \gamma_i (w_i - w^*) \nabla_w C(w_i) \ < \ \infty \tag{4.20}$$

We must introduce an additional condition on the learning rates $\gamma_i$. This condition limits the rate of decrease of the learning rates. Such a condition is required, because decreasing the learning rates too quickly could stop the progression of the algorithm towards the minimum. This condition is expressed by the divergence of the infinite sum of the learning rates:

$$\sum_{i=1}^{\infty} \gamma_i \ = \ \infty \tag{4.21}$$

Condition (4.21) is intuitively natural if we imagine that the current parameter is far away from the minimum in an area where the gradient is approximately constant. Successive updates $\gamma_t \nabla_w C(w_t)$ should be allowed to move the parameter to arbitrary distances.

Since we are dealing with positive quantities only, conditions (4.20) and (4.21) imply that:

$$(w_t - w^*) \nabla_w C(w_t) \ \underset{t\to\infty}{\longrightarrow} 0 \tag{4.22}$$

This result is similar to (4.5) and leads to the same conclusion about the convergence of the gradient descent algorithm.

$$w_t \ \underset{t\to\infty}{\longrightarrow} w^* \tag{4.23}$$

Besides the existence of a single minimum $w^*$ and the general convexity criterion (4.1), we had to introduce three additional conditions to obtain this convergence. Two conditions (4.15) and (4.21) directly address the learning rate schedule. The last condition (4.16) states that the growth of the gradients is limited.

## Comments

Condition (4.16) states that the gradient should not increase more than linearly when we move away from the minimum. Bounding the eigenvalues of the Hessian is an easy way to make sure that this condition holds. More general theorems however only require a polynomial bound on the size of the gradient (Benveniste, Metivier and Priouret, 1990).

The proof presented in this section addresses the case of *decreasing learning rates.* A different approach to step (b) leads to convergence results for the case of *constant learning rates.* Instead of bounding the second term of the variations (4.14) we can compare the sizes of both terms. Assuming condition (4.16) with $A = 0$, it appears that choosing a constant learning rate smaller than $\sqrt{2/B}$ makes the variations (4.14) negative. This result is consistent with the usual criterion since the minimal value of $B$ is the square of the highest eigenvalue of the Hessian matrix.

This analysis also provides convergence speed results: bounding the right hand side of (4.14) gives a measure of how quickly the Lyapunov sequence decreases. As expected, the best bounds are obtained when $(w_t - w^*)$ and $\gamma_t \nabla_w C(w_t)$ are aligned. This can be achieved by choosing a learning rate matrix $\gamma_t$ which approximates the inverse of the Hessian. Such a non scalar learning rates only introduces minor changes in the proofs. The learning rate matrix must be symmetric and definite positive. Conditions (4.15) and (4.21) then must refer to the highest and lowest eigenvalues of the learning rate matrix.

## 4.3 Lyapunov Process

Convergence proofs for the general online gradient algorithm (section 2.3) can be established using the same approach. It is obvious however that any online learning algorithm can be mislead by a consistent choice of very improbable examples. There is therefore no hope to prove that this algorithm always converges. The best possible result then is the *almost sure convergence*, that is to say that the algorithm converges towards the solution with probability 1.

Each iteration of the general gradient descent algorithm consists of drawing an event $\mathbf{z}_t$ from distribution $dP(z)$ and applying the update formula

$$w_{t+1} \;=\; w_t - \gamma_t H(\mathbf{z}_t, w_t) \tag{4.24}$$

where the update term $H(\mathbf{z}_t, w_t)$ fulfills the condition

$$\mathbf{E}_{\mathbf{z}}\, H(\mathbf{z}, w_t) \;=\; \nabla_w C(w_t) \tag{4.25}$$

and where the learning rates $\gamma_t$ are positive numbers or definite positive matrices. The main discussion in this section addresses scalar learning rates. Using a learning rate matrix introduces only minor changes discussed in the comments.

**Step a.** The first step in the proof consists in defining a *Lyapunov process* which measures how far we are from the solution.

$$h_t \;\stackrel{\triangle}{=}\; (w_t - w^*)^2. \tag{4.26}$$

Although definition (4.26) looks similar to the discrete batch gradient case (4.8), the notation $h_t$ in (4.26) denotes a random variable that depends on all the previous choices of example events $\mathbf{z}_t$.

**Step b.** As in the batch gradient case, an expression for the variations of $h_t$ can be derived using equations (4.24) and (4.26).

$$h_{t+1} - h_t \;=\; -2\gamma_t(w_t - w^*)H(\mathbf{z}_t, w_t) + \gamma_t^2(H(\mathbf{z}_t, w_t))^2 \tag{4.27}$$

The convergence proof for the discrete gradient descent (section 4.2.2) relies on lemma (4.13) to establish the convergence of the Lyapunov criterion. The lemma defines a sufficient condition based on the variations of the criterion. Expression (4.27) however explicitly refers to the random example $\mathbf{z}_t$. Using lemma (4.13) here would be an attempt to prove that the algorithm converges for all imaginable choice of the examples, including the most improbable, such as continuously drawing the same example.

The correct approach consists in removing this dependency by taking the conditional expectation of the variations (4.27) given all the information $\mathcal{P}_t$ that was available just before iteration $t$.

$$\mathcal{P}_t \;\stackrel{\triangle}{=}\; \mathbf{z}_o, \ldots, \mathbf{z}_{t-1},\;\; w_0, \ldots, w_t,\;\; \gamma_0, \ldots, \gamma_t \tag{4.28}$$

This conditional expectation of the variations gives sufficient information to apply the *quasi-martingale convergence theorem.*

## 4.4 Quasi-Martingales

The quasi-martingale convergence theorem is in fact very similar to the lemma (4.13) presented in section 4.2.2. The following discussion only presents the theorem without proof and exposes its analogy with this lemma. Proofs can be found in (Metivier, 1983) or (Fisk, 1965).

Given all the past information $\mathcal{P}_t$, we wish to define a deterministic criterion for distinguishing the "positive variations" from the "negative variations" of a process $u_t$. The sign of the variation $u_{t+1} - u_t$ is not an acceptable choice because it depends on $u_{t+1}$ which is not fully determined given $\mathcal{P}_t$. This problem can be solved by considering the conditional expectation of the variations.

$$\delta_t \;\stackrel{\triangle}{=}\; \begin{cases} 1 & \text{if } \mathbf{E}\left(u_{t+1} - u_t \,|\, \mathcal{P}_t\right) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.29}$$

The variable $\delta$ defined in (4.29) defines which variations are considered positive. The convergence of the infinite sum of the positive expected variations is a sufficient condition for the *almost sure convergence* of a positive process $u_t$.

$$\left.\begin{array}{c} \forall t,\; u_t \geq 0 \\ \displaystyle\sum_{t=1}^{\infty} \mathbf{E}(\delta_t(u_{t+1} - u_t)) < \infty \end{array}\right\} \Longrightarrow\; u_t \xrightarrow[t\to\infty]{a.s.} u_\infty \geq 0 \tag{4.30}$$

This result is a particular case of theorem 9.4 and proposition 9.5 in (Metivier, 1983). The name *quasi-martingale convergence theorem* comes from the fact that condition (4.30) also implies that the process $u_t$ is a quasi-martingale (Fisk, 1965). Comparing theorem (4.30) and lemma (4.13) explains easily why quasi-martingales are so useful for studying the convergence of online algorithms. This fact has been known since (Gladyshev, 1965).

## 4.5 Convergence of Online Algorithms (Convex Case)

This convergence result allow us to proceed with step (b) of our proof.

**Step b** (continued). The following expression is obtained by taking the conditional expectation of (4.27) and factoring the constant multipliers.

$$\begin{aligned} \mathbf{E}\left(h_{t+1} - h_t \mid \mathcal{P}_t\right) \;=\; & -2\,\gamma_t (w_t - w^*)\,\mathbf{E}\left(H(\mathbf{z}_t, w_t) \mid \mathcal{P}_t\right) \\ & + \gamma_t^2\,\mathbf{E}\left(H(\mathbf{z}_t, w_t)^2 \mid \mathcal{P}_t\right) \end{aligned} \tag{4.31}$$

This expression can be further simplified using condition (4.25).

$$\begin{aligned} &\mathbf{E}\left(h_{t+1} - h_t \mid \mathcal{P}_t\right) \\ &\quad= \; -2\,\gamma_t (w_t - w^*)\,\mathbf{E}_{\mathbf{z}}(H(\mathbf{z}, w_t)) + \;\gamma_t^2\,\mathbf{E}_{\mathbf{z}}(H(\mathbf{z}_t, w_t)^2) \\ &\quad= \; -2\,\gamma_t (w_t - w^*)\nabla_w C(w_t) + \;\gamma_t^2\,\mathbf{E}_{\mathbf{z}}(H(\mathbf{z}_t, w_t)^2) \end{aligned} \tag{4.32}$$

The first term of this upper bound is negative according to condition 4.1. As in section 4.2.2, two additional conditions are required to address the discrete dynamics of the algorithm. The first condition (4.33), similar to (4.15), states that the learning rates are decreasing fast enough.

$$\sum_{i=1}^{\infty} \gamma_t^2 < \infty \tag{4.33}$$

The second condition (4.34) serves the same purpose than condition (4.16). This term bounds the growth of the second moment of the update $H(\mathbf{z}, w)$.

$$\mathbf{E}_{\mathbf{z}}(H(\mathbf{z}, w)^2) \leq A + B(w - w^*)^2 \quad A, B \geq 0 \tag{4.34}$$

We can now transform equation (4.32) using this condition.

$$\mathbf{E}\left(h_{t+1} - (1 - \gamma_t^2 B) h_t \mid \mathcal{P}_t\right) \leq -2\gamma_t\,(w_t - w^*)\nabla_w C(w_t) \;+\; \gamma_t^2 A \tag{4.35}$$

We now define two auxiliary sequences $\mu_t$ and $h'_t$ as in (4.18). Multiplying both the left-hand-side and the right hand side of (4.32) by $\mu_t$, we obtain:

$$\mathbf{E}\left(h'_{t+1} - h'_t \mid \mathcal{P}_t\right) \leq \gamma_t^2 \mu_t A \tag{4.36}$$

A simple transformation then gives a bound for the positive expected variations of $h'_t$.

$$\mathbf{E}(\delta_t\,(h'_{t+1} - h'_t)) \;=\; \mathbf{E}(\delta_t\,\mathbf{E}\left(h'_{t+1} - h'_t \mid \mathcal{P}_t\right)\,) \;\leq\; \gamma_t^2 \mu_t A \tag{4.37}$$

Since this bound is the summand of a convergent infinite sum, theorem (4.30) implies that $h'_t$ converges almost surely. Since the sequence $\mu_t$ converges, the Lyapunov process $h_t$ also converges almost surely.

**Step c.** We now prove that the convergence of the Lyapunov process implies the convergence of the discrete gradient descent algorithm. Since $h_t$ converges, equation (4.35) implies the convergence of the following sum:

$$\sum_{i=1}^{\infty} \gamma_i (w_i - w^*) \nabla_w C(w_i) \;<\; \infty \quad \text{a.s.} \tag{4.38}$$

We must introduce an additional condition on the learning rates $\gamma_i$ which limits the rate of decrease of the learning rates. This condition is similar to condition (4.21).

$$\sum_{i=1}^{\infty} \gamma_i \;=\; \infty \tag{4.39}$$

Since we are dealing with positive quantities only, conditions (4.38) and (4.39) imply that:

$$(w_t - w^*) \nabla_w C(w_t) \;\underset{t\to\infty}{\xrightarrow{a.s.}}\; 0 \tag{4.40}$$

This result is similar to (4.5) or (4.22) and leads to the same conclusion about the convergence of the gradient descent algorithm.

$$w_t \;\underset{t\to\infty}{\xrightarrow{a.s.}}\; w^* \tag{4.41}$$

Besides the general convexity criterion (4.1), we had to introduce three additional conditions to obtain this convergence. Two conditions (4.33) and (4.39) directly address the learning rate schedule as in the batch gradient case. The last condition (4.34) is similar to condition (4.16) but contains an additional variance term which reflects the stochastic dynamics of the online gradient descent.

## Comments

Equations (4.14) and (4.32) look very similar. The second term of the right hand side of (4.32) however refers to the second moment of the updates instead of the norm of the gradients. This term can be decomposed as follows:

$$\gamma_t^2 \mathbf{E}_{\mathbf{z}}(H(\mathbf{z}, w))^2 \;=\; \gamma_t^2 (\nabla_w C(w))^2 \,+\, \gamma_t^2 \mathbf{var}_{\mathbf{z}} H(\mathbf{z}, w) \tag{4.42}$$

The second term of this decomposition depends on the noise implied by the stochastic nature of the algorithm. This variance remains strictly positive in general, even at the solution $w^*$. This fact is the main explanation for the dynamical differences between batch gradient descent and online gradient descent.

Let us assume that the algorithm converges. The first term of the right hand side of (4.32) tends towards zero, as well as the first term of (4.42). We can therefore write an asymptotic equivalent to the expected variation the Lyapunov criterion:

$$\mathbf{E}\left(h_{t+1} - h_t \mid \mathcal{P}_t\right) \underset{t\to\infty}{\asymp} \gamma_t\left(\gamma_t \mathbf{var_z} H(\mathbf{z}, w^*) - (w_t - w)\nabla_w C(w)\right) \qquad (4.43)$$

This result means that the quantities $\gamma_t \mathbf{var_z} H(\mathbf{z}, w^*)$ and $(w_t - w)\nabla_w C(w)$ keep the same order of magnitude during the convergence. Since the latter quantity is related to the distance to the optimum (cf. comments to section 4.2.2) the convergence speed depends on how fast the learning rates $\gamma_t$ decrease. This decrease rate is in turn limited by condition (4.39).

This analysis can be repeated with non scalar learning rates approximating the inverse of the Hessian. This algorithm converges faster than using a scalar learning rate equal to the inverse of the largest eigenvalue of the Hessian. This result of course assume that these learning rates still fulfill criterions (4.33) and (4.39), as in the batch gradient descent case (cf. comments to section 4.2.2).

The final comment expands section 3.2 discussing online gradient descent with non differentiable functions. The proof presented in this section never uses the fact that $\nabla_w C(w)$ is actually the gradient of the cost $C(w)$. All references to this gradient can be eliminated by merging conditions (4.1) and (4.25):

$$\forall\varepsilon > 0, \qquad \inf_{(w-w^*)^2>\varepsilon} (w - w^*)\,\mathbf{E}_z H(z, w) > 0 \qquad (4.44)$$

This condition (4.44), together with the usual conditions (4.33), (4.39) and (4.34), is sufficient to ensure the convergence of algorithm (4.24). This result makes no reference to a differentiable cost function.

## 5 General Online Optimization

This section analyzes the convergence of the general online gradient algorithm (section 2.3) without convexity hypothesis. In other words, the cost function $C(w)$ can now have several local minima.

There are two ways to handle this analysis. The first method consists of partitioning the parameter space into several attraction basins, discussing the conditions under which the algorithm confines the parameters $w_t$ in a single attraction basin, defining suitable Lyapunov criterions (Krasovskii, 1963), and proceeding as in the convex case. Since the online gradient descent algorithm never completely confines the parameter into a single attraction basin, we must also study how the algorithm hops from one attraction basin to another.

A much simpler method quickly gives a subtly different result. Instead of proving that the parameter $w_t$ converges, we prove the cost function $C(w_t)$ and its gradient $\nabla_w C(w_t)$ converge. The discussion presented below is an expanded version of the proof given in (Bottou, 1991).

## 5.1 Assumptions

The convergence results rely on the following assumptions:

i) The cost function $C(w)$ is three times differentiable with continuous derivatives. It is bounded from below, i.e. $C(w) \geq C_{\min}$. We can assume, without loss of generality, that $C(w) \geq 0$.

ii) The usual conditions on the learning rates are fulfilled.

$$\sum_{i=1}^{\infty} \gamma_t^2 < \infty, \quad \sum_{i=1}^{\infty} \gamma_t = \infty \tag{5.1}$$

iii) The second moment of the update term should not grow more than linearly with the size of the parameters. This condition is similar to (4.34).

$$\mathbf{E_z}(H(\mathbf{z}, w))^2 \leq A + Bw^2 \tag{5.2}$$

iv) When the norm of the parameter $w$ is larger than a certain horizon $D$, the opposite of the gradient $-\nabla_w C(w)$ points towards the origin.

$$\inf_{w^2 > D} w \nabla_w C(w) > 0 \tag{5.3}$$

v) When the norm of the parameter $w$ is smaller than a second horizon $E$ greater than $D$, the norm of the update term $H(\mathbf{z}, w)$ is bounded regardless of $\mathbf{z}$. This is usually a mild requirement.

$$\forall \mathbf{z}, \quad \sup_{w^2 < E} ||H(\mathbf{z}, w)|| \leq K_0 \tag{5.4}$$

Hypothesis (5.3) prevents the possibility of plateaus on which the parameter vector can grow indefinitely without ever escaping. Beyond a certain horizon, the update terms always moves $w_t$ closer to the origin on average.

This condition is easy to verify in the case of the $K$-Means algorithm (section 3.2.2) for instance. The cost function is never reduced by moving centroids beyond the envelope of the data points. Multi-layer networks (section 3.1.2) however do not always fulfill this condition because the sigmoid has flat asymptotes. In practice however, it is common to choose desired values that are smaller than the sigmoid asymptotes, and to add a small linear term to the sigmoid which makes sure that rounding errors will not make the sigmoid gradient negative. These well known tricks in fact ensure that condition (5.3) is fulfilled. A similar discussion applies to the LVQ2 algorithm (section 3.2.3).

## 5.2 Global Confinement

The first part of the analysis consists in taking advantage of hypothesis (5.3) and proving that the parameter vector $w_t$ is almost surely confined into a bounded region. The proof again relies on the same three steps.

**Step a.** We define a suitable criterion:

$$f_t \stackrel{\triangle}{=} \max(E, w_t^2) \tag{5.5}$$

**Step b.** The definition of $f_t$ implies that the variations of $f_t$ are bounded by the variations of $w_t^2$.

$$f_{t+1} - f_t \leq -2\gamma_t w_t H(\mathbf{z}_t, w_t) + \gamma_t^2 (H(\mathbf{z}_t, w_t))^2 \tag{5.6}$$

Inequality (5.6) is actually an equality when both $w_{t+1}^2$ and $w_t^2$ are greater than $E$. We can the write a bound for the expected variations:

$$\mathbf{E}\left(f_{t+1} - f_t \,|\, \mathcal{P}_t\right) \leq -2\gamma_t w_t \nabla_w C(w_t) + \gamma_t^2 \mathbf{E}_{\mathbf{z}} (H(\mathbf{z}, w_t))^2 \tag{5.7}$$

We can eliminate the first term of this bound by considering several cases:

- When both $w_{t+1}^2$ and $w_t^2$ are smaller than $E$, the variations $f_{t+1} - f_t$ are zero. The expected variations are therefore bounded by the second term of (5.7) which is positive.
- When $w_t^2$ is greater than $E$, hypothesis (5.3) ensures that the second term of the bound (5.7) is negative. We can safely remove this term.
- The remaining case has $w_t^2 < E$ and $w_{t+1}^2 \geq E$. The difference between $w_{t+1}$ and $w_t$ is $\gamma_t H(\mathbf{z}_t, w_t)$. Since hypothesis (5.4) ensures that $H(\mathbf{z}_t, w_t)$ is bounded, we can conclude that $w_t$ is greater than $D$ as soon as the decreasing learning rates become small enough. Invoking hypothesis (5.3) gives the final argument.

The following bound therefore is valid when $t$ is large enough:

$$\mathbf{E}\left(f_{t+1} - f_t \,|\, \mathcal{P}_t\right) \leq \gamma_t^2 \mathbf{E}_{\mathbf{z}} (H(\mathbf{z}, w_t))^2 \leq \gamma_t^2 (A + B f_t) \tag{5.8}$$

We now proceed along the well known lines. We first transform the bound on the expected variations as in (4.35). We define two auxiliary quantities $\mu_t$ and $f'_t$ as in (4.18). The expected variations of $f'_t$ are bounded as shown in equation (4.36). We can then bound the positive expected variations of $f'_t$.

$$\mathbf{E}(\delta_t (f'_{t+1} - f'_t)) \leq \mathbf{E}(\delta_t \mathbf{E}\left(f'_{t+1} - f'_t \,|\, \mathcal{P}_t\right)) \leq \gamma_t^2 \mu_t A \tag{5.9}$$

Theorem (4.30) then implies that $f'_t$ converges almost surely. This convergence implies that $f_t$ converges almost surely.

**Step c.** Let us assume that $f_t$ converge to a value $f_\infty$ greater than $E$. When $t$ is large enough, this convergence implies that both $w_t^2$ and $w_{t+1}^2$ are greater than $E$. Bound (5.6) is then an equality. This equality implies that the following infinite sum converges almost surely:

$$\sum_{i=1}^{\infty} \gamma_t w_t \nabla_w C(w_t) < \infty \quad \text{a.s.} \tag{5.10}$$

Since $\sum \gamma_t = \infty$ this result is not compatible with hypothesis (5.3). We must therefore conclude that $f_t$ converges to the smallest possible value $E$.

**Global confinement.** The convergence of $f_t$ means that the norm $w_t^2$ of the parameter vector $w_t$ is bounded. In other words, hypothesis (5.3) guarantees that the parameters will be confined in a bounded region containing the origin.

This confinement property means that all continuous functions of $w_t$ are bounded (we assume of course that the parameter space has finite dimension). This include $w_t^2$, $\mathbf{E}_{\mathbf{z}}(H(\mathbf{z}, w))^2$ and all the derivatives of the cost function $C(w_t)$. In the rest of this section, positive constants $K_1$, $K_2$, etc…are introduced whenever such a bound is used.

## 5.3 Convergence of Online Algorithms (General Case)

We now proceed with the analysis of the general online gradient algorithm. **Step a.** We define the following criterion:

$$h_t \overset{\triangle}{=} C(w_t) \geq 0 \tag{5.11}$$

**Step b.** We can then bound the variations of the criterion $h_t$ using a first order Taylor expansion and bounding the second derivatives with $K_1$.

$$|\, h_{t+1} - h_t + 2\gamma_t H(\mathbf{z}, w_t) \nabla_w C(w_t) \,| \leq \gamma_t^2 H(\mathbf{z}, w_t)^2 K_1 \quad \text{a.s.} \tag{5.12}$$

This inequality can be rewritten as:

$$h_{t+1} - h_t \leq -2\gamma_t H(\mathbf{z}, w_t) \nabla_w C(w_t) + \gamma_t^2 H(\mathbf{z}, w_t)^2 K_1 \quad \text{a.s.} \tag{5.13}$$

We now take the conditional expectation using (2.7):

$$\mathbf{E}\left(h_{t+1} - h_t \,|\, \mathcal{P}_t\right) \leq -2\gamma_t (\nabla_w C(w_t))^2 + \gamma_t^2 \mathbf{E}_{\mathbf{z}}(H(\mathbf{z}, w_t)) K_1 \tag{5.14}$$

This result leads to the following bound:

$$\mathbf{E}\left(h_{t+1} - h_t \,|\, \mathcal{P}_t\right) \leq \gamma_t^2 K_2 K_1 \tag{5.15}$$

The positive expected variations of $h_t$ are then bounded by

$$\mathbf{E}(\delta_t\,(h_{t+1}-h_t)) \;=\; \mathbf{E}(\delta_t\,\mathbf{E}\,(h_{t+1}-h_t \mid \mathcal{P}_t)\,) \;\leq\; \gamma_t^2 K_2 K_1 \qquad (5.16)$$

Since this bound is the summand of a convergent infinite sum, theorem (4.30) implies that $h_t = C(w_t)$ converges almost surely.

$$C(w_t) \;\xrightarrow[t\to\infty]{a.s} C_\infty \qquad (5.17)$$

**Step c.** The last step of the proof departs from the convex case. Proving that $C(w_t)$ converges to zero would be a very strong result, equivalent to proving the convergence to the global minimum. We can however prove that the gradient $\nabla_w C(w_t)$ converges to zero almost surely.

By taking the expectation of (5.14) and summing on $t = 1\ldots\infty$, we see that the convergence of $C(w_t)$ implies the convergence of the following infinite sum:

$$\sum_{t=1}^{\infty} \gamma_t(\nabla_w C(w_t))^2 < \infty \quad \text{a.s} \qquad (5.18)$$

This convergence does not imply yet that the squared gradient $\nabla_w C(w_t)$ converges. We now define a second criterion:

$$g_t \;\stackrel{\triangle}{=}\; (\nabla_w C(w_t))^2 \qquad (5.19)$$

The variations of $g_t$ are easily bounded using the Taylor expansion procedure demonstrated for the variations of $h_t$.

$$g_{t+1} - g_t \leq -2\gamma_t H(\mathbf{z},w)\nabla_w^2 C(w_t)\nabla_w C(w_t) + \gamma_t^2 (H(\mathbf{z},w)^2 K_3 \quad \text{a.s.} \qquad (5.20)$$

Taking the conditional expectation and bounding the second derivatives by $K_4$:

$$\mathbf{E}\,(g_{t+1}-g_t \mid \mathcal{P}_t) \leq 2\gamma_t(\nabla_w C(w_t))^2 K_4 + \gamma_t^2 K_2 K_3 \qquad (5.21)$$

We can then bound the positive expected variations of $g_t$:

$$\begin{aligned}\mathbf{E}(\delta_t(g_{t+1}-g_t)) &= \mathbf{E}(\delta_t \mathbf{E}\,(g_{t+1}-g_t \mid \mathcal{P}_t))\\ &\leq \gamma_t(\nabla_w C(w_t))^2 K_4 + \gamma_t^2 K_2 K_3\end{aligned} \qquad (5.22)$$

The two terms on the right hand side are the summands of convergent infinite sums (5.18) and (5.1). Theorem (4.30) then implies that $g_t$ converges almost surely. Result (5.18) implies that this limit must be zero.

$$g_t \xrightarrow[t\to\infty]{a.s} 0 \quad \text{and} \quad \nabla_w C(w_t) \xrightarrow[t\to\infty]{a.s} 0 \qquad (5.23)$$

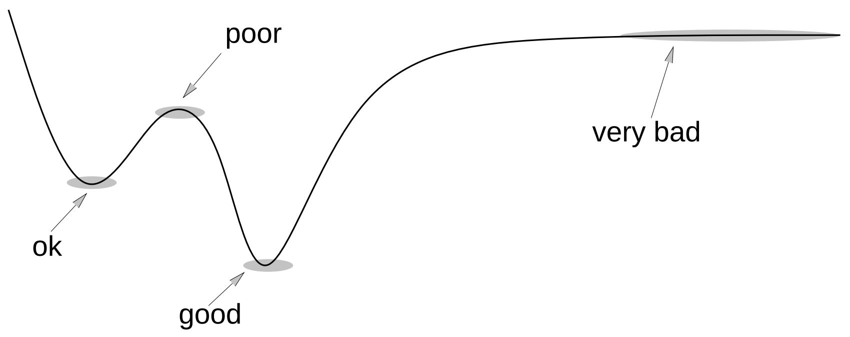

**Figure 8**: Extremal points include global and local minima. They also include poor solutions like saddle points and asymptotic plateaus. Every user of multi-layer network training algorithms is well aware of these possibilities.

## 5.4 Convergence to the Extremal Points

Let us summarize the convergence results obtained for the general gradient descent algorithm (section 2.3). These results are based on the five hypotheses presented in section 5.1.

i) The parameter vectors $w_t$ are *confined* with probability 1 in a bounded region of the parameter space. This result essentially is consequence of hypothesis (5.3).

ii) The cost function $C(w_t)$ converges almost surely.

$$C(w_t) \xrightarrow[t\to\infty]{a.s.} C_\infty$$

iii) The gradient $\nabla_w C(w_t)$ converges almost surely to 0.

$$\nabla_w C(w_t) \xrightarrow[t\to\infty]{a.s.} 0$$

The convergence of the gradient is the most informative result. Figure 8 shows several regions in which the gradient goes to zero. These regions include local minima, saddle points, local maxima and plateaus.

The confinement result prevents the parameter vector $w_t$ to diverge on an asymptotic plateau. Experience shows that hypothesis (5.3) is very significant. It is well known indeed that such a divergence occurs easily when the desired outputs of a multi-layer network are equal to the asymptotes of the sigmoid.

Saddle points and local maxima are usually unstable solutions. A small isotropic noise in the algorithm convergence can move the parameter vector away. We cannot however discard these solutions because it is easy to construct cases where the stochastic noise introduced by the online gradient descent procedure is not sufficient because it is not isotropic.

This *convergence to the extremal points* concludes our discussion.

## 6 Conclusion

The online learning framework presented in this document addresses a significant subset of the online learning algorithms, including, but not limited to, adaline, perceptron, multi-layer networks, k-means, learning vector quantization, and Kalman style algorithms. This formalism provides a clear statement of the goal of the learning procedure. It includes provisions for handling non-differentiable cost functions and quasi-Newton algorithms.

General convergence results are based on the theory of stochastic approximations. The main results address the convergence to the minimum of a convex cost function, and the convergence to the extremal points of a general cost function. The final convergence speed of online learning algorithm is amenable to a theoretical analysis using the same tools. The possibility of analyzing long range convergence speed, as achieved in restricted cases (Saad and Solla, 1996), remains an open question.

## Acknowledgments

I am very grateful to both Vladimir Vapnik and Yoshua Bengio. Their comments and suggestions resulted in many improvements to this document.

## References

Amari, S. (1967). A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, EC-16:299–307.

Becker, S. and LeCun, Y. (1989). Improving the Convergence of Back-Propagation Learning with Second-Order Methods. In Touretzky, D., Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 29–37, San Mateo. Morgan Kaufman.

Benveniste, A., Metivier, M., and Priouret, P. (1990). *Adaptive Algorithms and Stochastic Approximations*. Springer Verlag, Berlin, New York.

Bottou, L. (1991). *Une Approche théorique de l'Apprentissage Connexionniste: Applications à la Reconnaissance de la Parole*. PhD thesis, Université de Paris XI, Orsay, France.

Bottou, L. and Bengio, Y. (1995). Convergence Properties of the KMeans Algorithm. In *Advances in Neural Information Processing Systems*, volume 7, Denver. MIT Press.

Dennis, J. and Schnabel, R. B. (1983). *Numerical Methods For Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Fisk, D. (1965). Quasi-martingales. *Transactions of the American Mathematical Society*, (120):359–388.

Gladyshev, E. (1965). On stochastic approximations. *Theory of Probability and its Applications*, 10:275–278.

Hebb, D. O. (1949). *The Organization of Behavior.* Wiley, New York.

Kohonen, T. (1982). Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43:59–69.

Kohonen, T., Barna, G., and Chrisley, R. (1988). 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.

Krasovskii, A. A. (1963). *Dynamic of continuous self-Organizing Systems.* Fizmatgiz, Moscow. (in russian).

Kushner, H. J. and Clark, D. S. (1978). *Stochastic Approximation for Constrained and Unconstrained Systems.* Applied Math. Sci. 26. Springer Verlag, Berlin, New York.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551.

Ljung, L. and Söderström, T. (1983). *Theory and Practice of recursive identification.* MIT Press, Cambridge, MA.

MacQueen, J. (1967). Some Methods for Classification and Analysis of Multivariate Observations. In LeCam, L. M. and Neyman, J., editors, *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics, and Probabilities*, volume 1, pages 281–297, Berkeley and Los Angeles, (Calif). University of California Press.

Mendel, J. M. and Fu, K. S. (1970). *Adaptive, Learning, and Pattern Recognition Systems: Theory and Applications.* Academic Press, New York.

Metivier, M. (1981). Martingale et convergence p.s. d'algorithmes stochastiques. In *Outils et modèles mathématiques pour l'automatique et le traitement du signal*, volume 1, pages 529–552. Editions du CNRS, Paris, France.

Metivier, M. (1983). *Semi-Martingales.* Walter de Gruyter, Berlin.

Minsky, M. and Papert, S. (1969). *Perceptrons.* MIT Press, Cambridge, MA.

Müller, U., Gunzinger, A., and Guggenbühl, W. (1995). Fast neural net simulation with a DSP processor array. *IEEE Trans. on Neural Networks*, 6(1):203–213.

Robbins, H. and Monro, S. (1951). A Stochastic Approximation Model. *Ann. Math. Stat.*, 22:400–407.

Rosenblatt, F. (1957). The Perceptron: A perceiving and recognizing automaton. Technical Report 85-460-1, Project PARA, Cornell Aeronautical Lab.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). 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.

Saad, D. and Solla, S. A. (1996). Dynamics of On-Line Gradient Descent Learning for Multilayer Neural Networks. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 302–308, Cambridge, MA. MIT Press.

Schumann, M. and Retzko, R. (1995). Self Organizing Maps for Vehicle Routing Problems - minimizing an explicit cost function. In Fogelman-Soulié, F. and Gallinari, P., editors, *Proc. ICANN'95, Int. Conf. on Artificial Neural Networks*, volume II, pages 401–406, Nanterre, France. EC2.

Tsypkin, Y. (1971). *Adaptation and Learning in automatic systems.* Academic Press, New York.

Tsypkin, Y. (1973). *Foundations of the theory of learning systems.* Academic Press, New York.

Vapnik, V. N. (1982). *Estimation of dependences based on empirical data.* Springer Series in Statistics. Springer Verlag, Berlin, New York.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory.* Springer Verlag, Berlin, New York.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *IRE WESCON Conv. Record, Part 4.*, pages 96–104.

Widrow, B. and Stearns, S. D. (1985). *Adaptive Signal Processing.* Prentice-Hall.