International Institute for Applied Systems Analysis
www.iiasa.ac.at

Hebbian Rules in Neural Networks

Aubin, J.-P.

IIASA Working Paper

WP-94-050

June 1994

Aubin J-P (1994). Hebbian Rules in Neural Networks. IIASA Working Paper. IIASA, Laxenburg, Austria: WP-94-050
Copyright © 1994 by the author(s). http://pure.iiasa.ac.at/id/eprint/4160/

Working Papers on work of the International Institute for Applied Systems Analysis receive only limited review. Views or opinions expressed herein do not necessarily represent those of the Institute, its National Member Organizations, or other organizations supporting the work. All rights reserved. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage. All copies must bear this notice and the full citation on the first page. For other purposes, to republish, to post on servers or to redistribute to lists, permission must be sought by contacting repository@iiasa.ac.at

Working Paper

Hebbian Rules in Neural Networks

Jean-Pierre Aubin

WP-94-50
June 1994

IIASA International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 71521 □ Telex: 079 137 iiasa a □ Telefax: +43 2236 71313

Hebbian Rules in Neural Networks

Jean-Pierre Aubin

WP-94-50
June, 1994

CEREMADE, Université de Paris-Dauphine & IIASA

Working Papers are interim reports on work of the International Institute for Applied Systems Analysis and have received only limited review. Views or opinions expressed herein do not necessarily represent those of the Institute or of its National Member Organizations.

IIASA International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 71521 □ Telex: 079 137 iiasa a □ Telefax: +43 2236 71313

Hebbian Rules in Neural Networks

Jean-Pierre Aubin

CEREMADE, Université de Paris-Dauphine
&
IIASA, International Institute for Applied Systems Analysis

FOREWORD

Control theory allows us to present in a unified way many examples of neural networks by regarding them as discrete or continuous control systems, the states of which are the **signals** and the controls are the **synaptic matrices** (pattern classification problems, including time series in forecasting). We refer to (Aubin J.-P., 1994) and its bibliography for an approach using this viewpoint.

The usual classification problem for which most neural networks have been designed is then to find synaptic matrices with which the network maps **inputs** of a prescribed sequence of **patterns** to **outputs**, through one, several or a continuous set of **layers**. One can use several results on control of nonlinear systems to obtain **learning algorithms** converging to some solutions of this problem, such as the **back-propagation formula** (which is nothing other than the **adjoint equation** in control theory). The mathematical structure of the space of synaptic matrices involving tensor products, we then obtain in the formulas describing the learning algorithms **pure tensor products**, whose entries are the products of the presynaptic and postsynaptic activities. This explains in a mathematical way the emergence of **Hebbian rules** in learning mechanisms of neural networks.

Hebbian Rules in Neural Networks

Jean-Pierre Aubin

1 Adaptive Systems

The general form of an adaptive network is given by a map $\Phi : X \times U \to Y$ where X is the input space, Y the output space and U a control space (or parameter space). In this paper, the spaces X, Y and U are finite dimensional vector spaces.

A pattern is an "input-output" pair $(a_p, b_p) \in X \times Y$. The inputs a_p are often called keys (or search argument) and the outputs b_p the memorized data.

In the case of neural networks, or more generally, of connectionist networks, the control space is a matrix space (synaptic matrices, fitness matrices, activation matrices, etc., depending on the contexts and the fields of motivation). A pattern $(x, y) \in X \times Y$ made of an input-output pair is recognized (or discovered, generalized) by the adaptive system programmed by such a control u if $y = \Phi(x, u)$ is a signal processed by the network excited by the input x.

1.1 The Learning Problem

A teaching set $\mathcal{P}$ is a finite set of patterns $(a_p, b_p)_{p \in \mathcal{P}} \subset \mathcal{K}$ of input-output signals.

The choice of a control is done by learning a given number of input-output pairs (a_p, b_p) $(p \in \mathcal{P})$, (the teaching set), i.e., by finding a control $u_{\mathcal{P}}$ satisfying:

$$\Phi(a_p, u_{\mathcal{P}}) = b_p \text{ for all } p \in \mathcal{P} \tag{1}$$

We shall say that such a control $u_{\mathcal{P}}$ has learned the teaching set. With such a control u, the system generalizes from this teaching set by associating with any input x the output $y = \Phi(x, u_{\mathcal{P}})$. This is called the generalization phase of the operation of the adaptive system.

This includes the forecasting problem, when $X = Y$ and when the patterns associated with a time series $a_1, \ldots, a_{T+1}$ are defined by $a_p := a_t$ and $b_p := a_{t+1}$ (the input is the present state and the output the future state).

$$\Phi(a_t, u_{\mathcal{P}}) = a_{t+1} \text{ for all } t = 1, \ldots, T$$

The record of past states constitutes the teaching set.

A common feature to adaptive systems is that they are not "programmed" by a sequence of instructions, but by making a control u (a synaptic matrix W in the case of neural networks) learn a set of patterns which they can reproduce, and thus, hopefully, discover new patterns associated with such a synaptic matrix. It is nothing else, after all, than an extrapolation procedure set in a new framework.

Any algorithm converging to a solution $u_{\mathcal{P}}$ to the learning problem (??) is regarded as a learning algorithm.

Since the problem (??) is generally a nonlinear problem, the first algorithm we can think of is the Newton Algorithm. However, we can view the learning problem as an optimization problems and use familiar gradient algorithms.

1.2 The Gradient Algorithms

We observe that the learning problem (1) is a nonlinear problem, which can be mapped to a minimization problem of the form

$$0 = \inf_u \left(\sum_{p \in \mathcal{P}} E_p(\Phi(a_p, u) - b_p)^\alpha \right)^{1/\alpha}$$

where $\alpha \in [1, \infty]$ and where $E_p : Y \to \mathbf{R}$ are evaluation functions vanishing (only) at the origin.

There are two advantages for doing that. The first one is the possibility of defining a solution to such a minimization problem even when there is no solution to the above system of equations: When the infimum is not equal to 0, the minimal solution can be regarded as a quasi-solution. The second one is the possibility of using the whole family of minimization algorithms (including variants of the gradient method) converging to a solution to the minimization problem.

The simplest example of distance is naturally provided by the case when $\alpha = 2$. But we also include the important (although, nonsmooth) case when $p = \infty$, where we have to solve the minimization problem:

$$0 = \inf_u \sup_{p \in \mathcal{P}} E_p(\Phi(a_p, u) - b_p)$$

event though the functional is no longer differentiable in the usual sense.

Once we have transformed the learning problem into an optimization problem of the form $\inf_{u \in U} H(u)$ the natural algorithms which may lead to the minima are in most cases variants of the Gradient Algorithm. When H is differentiable, it can be written

$$u^{n+1} - u^n = -\varepsilon_n H'(u^n)$$

When the criterion is nonsmooth, the gradient of the criterion is replaced by a "generalized gradient". This happens when $\alpha = \infty$ in the above criteria, when the function to be minimized is written $H(u) := \sup_{p \in P} H_p(u)$. Even when the functions H_p are differentiable, taking their supremum destroys the usual differentiability. If $P(u)$ denotes the set of indices $p \in P$ achieving the maximum (i.e., the set of indices of "active" functions H_p such that $H(u) = H_p(u)$), then the generalized gradient is given by

$$\partial H(u) = \left\{ \sum_{p \in P(u)} \lambda_p H_p'(u) \right\}_{\lambda_p \geq 0, \sum_{p \in P(u)} \lambda_p = 1}$$

(For instance, the generalized gradient of $u \to |u| = \sup_{-u,u}$ at 0 is the interval $[-1, +1]$). In general, one defines in nonsmooth analysis the concept of the generalized gradient of H at u, which is generally a subset $\partial H(u)$.

In this case, the gradient algorithm takes the form

$$u^{n+1} - u^n \in -\varepsilon_n \partial H(u^n)$$

The convergence of these algorithms holds true under convexity assumptions both in the smooth and nonsmooth cases.

1.3 Learning without Forgetting

An algorithm which learns without forgetting is defined as follows. Assume that u_P has been obtained for learning the teaching set of $(a_p, b_p)_{1 \leq p \leq P-1}$ made of $P-1$ input-output pairs. At the Pth iteration, we add another pair (a_P, b_P) to the teaching set. We want to find a control u_P which learns the whole new teaching set $(a_p, b_p)_{1 \leq p \leq P}$ (and not only the last pattern).

A natural way to design such an algorithm is to use the control u_{P-1} which has learned the teaching set $(a_p, b_p)_{1 \leq p \leq P-1}$ and to modify it as little as possible for learning the new teaching set $(a_p, b_p)_{1 \leq p \leq P}$: this is what we call the heavy algorithm. Formally, u_P is obtained from u_{P-1} in the following way:

$$\begin{cases} u_P \text{ minimizes } u \to \|u - u_{P-1}\| \\ \\ \text{under the constraints: } \Phi(a_p, u) \;=\; b_p \text{ for all } t = 1, \ldots, P, \end{cases}$$

When the system is affine with respect to the control (but still nonlinear with respect to the state), the Heavy Algorithm provides very simple formulas, as we shall see in the specific case of neural networks.

2 Neural Networks

In this paper, a neural network denotes a adaptive system where the set of controls are (synaptic) matrices, or sequences of (synaptic) matrices or even time dependent synaptic matrices.

Before being more specific about examples of neural networks, we posit the question: What is then special to neural networks?

We shall have to apply the algorithms (Newton's Algorithm, Gradient Algorithms, Heavy Algorithms) we mentioned to adaptive systems controlled by synaptic matrices, and thus, to use specific properties of the spaces of matrices (regarded as tensor products).

Since Hebb's 1949 classic book ORGANIZATION OF BEHAVIOR, most of the studies of neural networks deal with numerous variations of learning rules which prescribe a priori the evolution of the synaptic matrix $W^n = (w^n_{i,j})$ through an algorithm of the form:

$$w^{n+1}_{i,j} - w^n_{i,j} = \alpha^n_i \beta^n_j$$

where the correction of the synaptic weight $w^n_{i,j}$ is proportional to the product of presynaptic and postsynaptic activity.

We shall see that the algorithms mentioned above, when applied to neural networks, i.e., adaptive systems parametrized by synaptic matrices, yield Hebbian Learning Rules, since they involve tensor products of vectors.

2.1 Classes of Neural Networks

For simplicity, we consider the learning problem of one pattern only.

In the case of one-layer neural network described by a propagation function $g : Y \to Y$, the map Φ is defined by

$$\Phi(a, W) := g(Wa)$$

where the input $a \in X$ and the output $b \in Y$ of a pattern (a, b) is given. Given an evaluation function E on the output space Y and an pattern (a, b), we look for a synaptic matrix which minimizes the function

$$W \to H(W) := E(g(Wa) - b)$$

In the case of multi-layer neural network described by the vector spaces (layers) $X_0 := X$, X_l $(l = 1, \ldots, L-1)$ and $X_L := Y$ and propagation rules $g_l : X_l \to X_l$ from one layer to the next, the map Φ_L associates with a the sequence $(W_1, \ldots, W_L)$ of synaptic matrices the final state x_L of the sequence of states starting from $x_0 := a$ according to

$$x_l = g_l(W_l x_{l-1}) \text{ for all } l = 1, \ldots L \tag{2}$$

We can regard a multi-layer neural network as a discrete control problem where layers play the role of time. At time (layer) 1, one chooses the control W_1 for mapping the initial state x_0 to $g_1(W_1 x_0)$, and so on: at time (layer) l, we associate with the preceding state x_{l-1} and the control W_l the new state $g_l(W_l x_{l-1})$. Hence the final state x_L is obtained from the initial state x_0 through the discrete dynamical system (2) controlled by the sequence of synaptic matrices W_l.

"Continuous-layer" neural networks are neural networks in which the evolution of the signals is governed by a differential equation

$$x'(t) = g(W(t)x(t)) \tag{3}$$

(where the time here is regarded as an "infinitesimal layer"). We denote by $\Phi_T(a, W(\cdot))$ the map which associates with a time-dependent synaptic matrix $W(t)$ and an initial signal a the final state $x(T)$ obtained through this control system.

Hence, we have to find a time-dependent synaptic matrix $W(\cdot)$ minimizing the function

$$W(\cdot) \to H(W(\cdot)) := E(\Phi_T(a, W(\cdot)) - b)$$

Again, in the continuous time version, we see that the final state x_T is obtained from the initial state x_0 through the dynamical system (3) controlled by the synaptic matrices $W(t)$.

In each of these cases, we have associated an adaptive system controlled by synaptic matrices. Devising gradient algorithms for minimizing such functions (or other algorithms) require the computation of the gradients of such functions defined on spaces of matrices. This is where the tensor structure of the spaces of linear operators pops up, since such gradients involve tensor products of vectors, which yield Hebbian rules, and justify the specific role of neural networks compared to general adaptive systems. Furthermore, since multi-layer neural networks are controlled problems, we can apply

the derivatives of these functions which involve the adjoint equations of the linearized dynamical systems

$$p_l = W_{l+1}^{n\star} g'_{l+1}(W_l x_l)^\star p_{l+1}$$

retropropagating the gradient $E'(x_L - b)$ of the error on the final state and

$$p'(t) = -W(t)^\star g'(W(t)x(t))^\star p(t)$$

retropropagating the gradient $E'(x(T) - b)$ of the error on the final state.

These two facts allow us to derive the back-propagation of the gradients formula.

Hence the task at hand is to compute the derivatives of these maps Φ and their transpose, in order to write down the algorithms.

3 Tensor Products

3.1 Tensor Products of Vectors

Let $X := \mathbf{R}^n$be a fundamental, $X^\star := \mathcal{L}(X, \mathbf{R})$ denote the **dual space** of X, which is the vector space of linear functionals $p : X \to \mathbf{R}$ on X. The bilinear form $< p, x >:= p(x)$ on $X^\star \times X$ is called the **duality product**. Let $Y := \mathbf{R}^{\mathbf{m}}$ be another finite dimensional vector space. We recall that the transpose $W^\star \in \mathcal{L}(Y^\star, X^\star)$ is defined by

$$< W^\star q, x > = < q, Wx > \quad \text{for all } x \in X, \quad \text{for all } q \in Y^\star$$

Let p belong to $X^\star$. We shall identify the linear form $p \in \mathcal{L}(X, \mathbf{R}) : x \in X \to < p, x >$ with its transpose $p \in \mathcal{L}(\mathbf{R}, X^\star) : \lambda \to \lambda p \in X^\star$. In the same way, we shall identify $x \in X$ with both the maps $p \in X^\star \to < p, x >$ and $\lambda \in \mathbf{R} \to \lambda x \in X$.

We associate with any $p \in X^\star$ and $y \in Y$ their **tensor product** $p \otimes y \in \mathcal{L}(X, Y)$, *which is the linear operator defined by*

$$p \otimes y \ : x \ \to \ (p \otimes y)(x) \ := < p, x > y$$

the matrix of which is

$$\left(p^i y_j\right)_{\substack{i=1,\dots,m \\ j=1,\dots,n}}$$

Its transpose $(p \otimes y)^\star \in \mathcal{L}(Y^\star, X^\star)$ maps $q \in Y^\star$ to

$$(p \otimes y)^\star(q) =< q, y > p$$

because $< q, < p, x > y >=<< q, y > p, x >$. Hence, **we shall identify from now on the transpose** $(p \otimes y)^\star$ **with** $y \otimes p$.

In the same way, if $q \in Y^\star$ and $x \in X$ are given, the tensor product $q \otimes x$ denotes the map $y \in Y \to < q, y > x \in X$ belonging to $\mathcal{L}(Y, X)$ and its transpose $p \in X^\star \to < p, x > q \in Y^\star$ belonging to $\mathcal{L}(X^\star, Y^\star)$ is identified with $x \otimes q$.

We observe that the map $(p, y) \in X^\star \times Y \to p \otimes y \in \mathcal{L}(X, Y)$ is bilinear.

It is useful to set

$$X^\star \otimes Y \ := \ \mathcal{L}(X, Y)$$

to emphasize this structure.

3.2 Tensor Products of Linear Operators

Let us consider two pairs (X, X_1) and (Y, Y_1) of finite dimensional vector-spaces. Let $A \in \mathcal{L}(X_1, X)$ and $B \in \mathcal{L}(Y, Y_1)$ be given. Since $A^\star \in \mathcal{L}(X^\star, X_1^\star)$, we denote by $A^\star \otimes B$ the linear operator from $X^\star \otimes Y := \mathcal{L}(X, Y)$ to $X_1^\star \otimes Y_1 := \mathcal{L}(X_1, Y_1)$ defined by

$$(A^\star \otimes B)(W) \ := \ BWA \ \text{ for all } \ W \in \mathcal{L}(X, Y)$$

We observe that when $W = p \otimes y$, we have

$$(A^\star \otimes B)(p \otimes y) = A^\star p \otimes By$$

and that

$$(A^\star \otimes B)^\star \ = \ A \otimes B^\star \tag{4}$$

Let $A_1 \in \mathcal{L}(X_2, X_1)$ and $B_1 \in \mathcal{L}(Y_1, Y_2)$. Then, it is easy to check that

$$(A_1^\star \otimes B_1)(A^\star \otimes B) \ = \ (A_1^\star A^\star) \otimes (B_1 B)$$

3.3 Gradients of Functionals on Linear Operators

One main reason to introduce the concepts of tensor products is given by the following

Proposition 3.1 *Let us consider three spaces X, Y and Z, an element $x \in X$, a differential map $g : Y \to Z$, with which we associate the map $\Psi : W \to \Psi(W) := g(Wx)$.*

The derivative of Ψ at $W \in \mathcal{L}(X, Y)$ is given by

$$\Psi'(W) \ = \ x \otimes g'(Wx)$$

Furthermore, if E be a differentiable functional from Z to $\mathbf{R}$, setting $H(W) := E(g(Wx))$, the gradient of H at $W \in \mathcal{L}(X, Y)$ is given by

$$H'(W) \ = \ x \otimes g'(Wx)^\star E'(g(Wx)) \in \mathcal{L}(X^\star, Y^\star)$$

Proof — Indeed,

$$\begin{cases} \displaystyle\lim_{\lambda \to 0} \frac{\Psi(W + \lambda U) - \Psi(W)}{\lambda} \\ \displaystyle = \lim_{\lambda \to 0} \frac{g(Wx + \lambda Ux) - g(Wx)}{\lambda} \\ = \ g'(Wx)Ux \ = \ (x \otimes g'(Wx))(U) \end{cases}$$

The chain rule implies the second formula.

4 Back-Propagation Algorithms

4.1 Gradient Methods

Multi-layer and continuous-layer neural networks being special discrete and continuous control problems, we can derive from control theory the formulas for the derivatives of the maps Φ_L and Φ_T and their transposes involving the "adjoint equation". The formulas on tensor products are then used when the controls are synaptic matrices.

In the case of a one-layer neural network described by a propagation function $g : Y \to Y$, the map Φ is defined by

$$\Phi(a, W) \;:=\; g(Wa)$$

and given an evaluation function E on the output space Y and a pattern (a, b), we look for a synaptic matrix which minimizes the function

$$W \;\to\; H(W) \;:=\; E(g(Wa) - b)$$

Then the gradient of H is given by

$$H'(W) \;=\; a \otimes g'(Wa)^{\star} E'(g(Wa) - b)$$

the entries of which are equal to

$$\frac{\partial H}{\partial w_{ij}} = a_j \left(\sum_{k=1}^{m} \frac{\partial g_k}{\partial y_i}(Wa) \frac{\partial E}{\partial y_k}(g(Wa) - b) \right)$$

The gradient method can then be written

$$W^{n+1} - W^n \;=\; -\varepsilon_n a \otimes g'(W^n a)^{\star} E'(g(W^n a) - b)$$

It yields for each entry:

$$w_{ij}^{n+1} = w_{ij}^{n} - \varepsilon_n a_j \left(\sum_{k=1}^{m} \frac{\partial g_k}{\partial y_i}(W^n a) \frac{\partial E}{\partial y_k}(g(W^n a) - b) \right)$$

It belongs to the class of *Hebbian learning rules*: the synaptic weight from a neuron j to a neuron i should be *strengthened* whenever the connection is highly active, in proportion of the activity of the pre-synaptic and post-synaptic neurons of the synapse.

In the case of a multi-layer network described by the vector spaces $X_0 := X$, X_l $(l = 1, \ldots, L-1)$ and $X_L := Y$ and differentiable propagation rules $g_l : X_l \to X_l$, the map Φ_L associates with a the sequence $(W_1, \ldots, W_L)$ of synaptic matrices the final state x_L of the sequence of states starting from $x_0 := a$ according to

$$x_l \;=\; g_l(W_l x_{l-1}) \;\text{ for all }\; l = 1, \ldots L$$

Given an evaluation function E defined on the output space X_L, we are looking for a sequence $(W_l)_{l=1,\ldots,L}$ of synaptic matrices minimizing the function

$$(W_1, \ldots, W_L) \to H(W_1, \ldots, W_L) \;:=\; E(\Phi_L(a, W_1, \ldots, W_L) - b)$$

The "back-propagation" learning rule is nothing else than the gradient method applied to this function H:

When the maps g_l and the evaluation function E are differentiable, the gradient of H is given by the formula

$$H'(W) \;=\; (x_{l-1} \otimes g_l'(x_l)^\star p_l)_{1 \le l \le L}$$

where p_l is given by the adjoint equation

$$p_l \;=\; W_{l+1}^{n\star} g_{l+1}'(W_l x_l)^\star p_{l+1}$$

retropropagating the gradient $E'(x_L - b)$ of the error on the final state. We check that

$$p_l = W_{l+1}^\star g_{l+1}'(x_{l+1})^\star \cdots W_L^\star g_L'(x_L)^\star E'(x_L - b)$$

In other words,

$$\frac{\partial H(W)}{\partial W_l} = x_{l-1} \otimes g_l'(x_l)^\star p_l$$

The gradient method provides the celebrated **back-propagation algorithm**: Starting with a synaptic matrix $\vec{W}^0$, we define $\vec{W}^{n+1}$ from the synaptic matrix $\vec{W}^n$ according to the rule

$$W_l^{n+1} - W_l^n \;=\; -\varepsilon_n x_{l-1}^n \otimes g_l'(x_l^n)^\star p_{l+1}^n$$

where $x_l^n = g_l(W_l^n x_{l-1}^n)$ (starting at $x_0^n = a$) and where p_l^n is obtained from the final condition $E'(x_L^n - b)$ through the adjoint equation of the linearized discrete control system:

$$p_l^n \;=\; W_{l+1}^{n\star} g_{l+1}'(x_{l+1}^n)^\star p_{l+1}^n$$

It is given by

$$p_l^n \;=\; W_{l+1}^{n\star} g_{l+1}'(x_{l+1}^n)^\star p_{l+1}^n \cdots W_L^{n\star} g_L'(x_L^n)^\star E'(x_L^n - b)$$

We begin by modifying the synaptic weights of the synaptic matrix of the last layer L by setting $p_L^n := E'(x_L^n - b)$ and then computing $p^{L-1} := W_L^{n\star} g_L'(x_L^n)^\star p_L^n$: the new matrix is obtained through the **Hebbian rule**

$$W_L^{n+1} - W_L^n \;=\; -\varepsilon_n x_{L-1}^n \otimes g_L'(W_L^n x_L^n)^\star p_L^n$$

Then, the gradient is "back-propagated" to modify successively the synaptic matrices W_l^n of each layer l from the last one to the first one through the adjoint equation and, at each layer, according to an Hebbian rule.

In the case of a continuous-layer network, the evolution of the signals is governed by

$$x'(t) \;=\; g(W(t)x(t))$$

Denoting by $\Phi_T(a, W(\cdot))$ the map which associates with a time-dependent synaptic matrix $W(t)$ and an initial signal a the final state $x(T)$ obtained through this control system, we have to minimize the function

$$W(\cdot) \to H(W(\cdot)) \;:=\; E(\Phi_T(a, W(\cdot)) - b)$$

The *continuous back-propagation* learning rule is nothing else than the gradient method applied to this function H.

We associate with a time-dependent synaptic matrix $W(\cdot)$ the solution $x(\cdot)$ to the continuous neural network starting at $x(0) = a$ the solution $p(\cdot)$ to the adjoint differential equation

$$p'(t) = -W(t)^\star g'(W(t)x(t))^\star p(t)$$

starting at $p_W(T) = E'(x(T) - b)$. Then the gradient of H is

$$H'(W(\cdot))(t) = x(t) \otimes g'(W(t)x(t))^\star p_W(t)$$

The gradient method provides the continuous back-propagation algorithm. Starting with a synaptic matrix $W^0(\cdot)$, we define $W^{n+1}(\cdot)$ from the synaptic matrix $W^n(\cdot)$ according to the rule: For all $t \in [0, T]$,

$$W^{n+1}(t) - W^n(t) = -\varepsilon_n x^n(t) \otimes g'(W^n(t)x^n(t))^\star p^n(t)$$

where $x^n(\cdot)$ is the solution to the differential equation

$$\frac{d}{dt}x^n(t) = g(W^n(t)x^n(t))$$

starting at $x^n(0) = a$ and where $p^n(\cdot)$ is the solution to the adjoint differential equation

$$\frac{d}{dt}p^n(t) = -W^n(t)^\star g'(W^n(t)x^n(t))^\star p^n(t)$$

starting at $p^n(T) = E'(x^n(T) - b)$.

4.2 The need for nonsmooth optimization

We may not always have the possibility to choose the evaluation function E and for instance, the opportunity to choose the simplest ones, as in the case of *least-square methods*. In the learning problem of control systems, for instance, when one deals with viable solutions to control systems, the evaluation function is the function measuring the distance from a point to a set, which is not differentiable. However, nonsmooth analysis, and in particular, convex analysis, provide ways to define *generalized gradients* of any function. Generalized gradient (which boil down to *subdifferential* in the case of convex functions) are no longer elements, but subsets. They allow us to extend the gradient methods. This is not the right place to recall precisely how generalized gradients are rigorously defined: We refer to (Aubin J.-P., 1993) or to Chapter 6 of (Aubin J.-P. & Frankowska H., 1990) for an introduction to nonsmooth analysis. For instance, if E is locally Lipschitz and g is differentiable, we deduce that the generalized gradient of the function H is given by

$$\partial H(W) = a \otimes g'(Wa)^\star \partial E(g(Wa) - b)$$

since the generalized gradient of the function $\Psi : y \to \Psi(y) := E(g(y))$ is equal to $g'(y)^\star \partial E(g(y))$.

We refer to the papers by Nicolas Seube and to (Aubin J.-P., 1994) for further details on this topic.

5 The Heavy Learning Algorithm for Neural Networks

We consider a class of neural networks of the form

$$\Phi(x, W) \;:=\; c(x) + G(x)W\psi(x)$$

where

$$\begin{cases} i) & \psi : X \to Z_1 \text{ maps the input space to the input layer} \\ ii) & \text{the synaptic matrix } W \in \mathcal{L}(Z_1, Z_2) \\ & \text{weights the processed signal } \psi(x) \\ iii) & G : X \to \mathcal{L}(Z_2, Y) \text{ maps the weighted processed signal to} \\ & \text{the output space} \end{cases}$$

which are affine with respect to the synaptic matrices. Let $\mathcal{P}$ be a finite teaching set of patterns $(a_p, b_p) \in X \times Y$. Hence we have to find a synaptic matrix W_P learning the training set in the sense that

$$c(a_p) + G(a_p)W_P\psi(a_p) \;=\; b_p \text{ for all } p = 1, \ldots, P \tag{5}$$

Assuming that W_{P-1} has been obtained for learning the $P-1$ input-output pairs $(a_p, b_p)_{1 \le p \le P-1}$, we want to find a new synaptic matrix W_P which learns the whole new teaching set $(a_p, b_p)_{1 \le p \le P}$ according to the "heavy rule"

$$W_P \text{ minimizes } W \to \|W - W_{P-1}\|$$

under the constraints (5).

For that purpose, we need the concept of pseudo-inverse $C^{\ominus 1} \in \mathcal{L}(Y, X)$ of a matrix $C \in \mathcal{L}(X, Y)$. It maps any $y \in Y$ to the closest solution with minimal norm $\bar{x} = C^{\ominus 1}y$, i.e., the solution with minimal norm to the equation $Cx = \bar{y}$ where $\bar{y}$ is the projection of y onto the image of C. The pseudo-inverse coincides with the usual inverse when C is invertible, is involved in quadratic minimization problems under linear equality constraints and enjoys many properties (which are often used in statistical analysis).

Since we define the synaptic matrix W_P as a solution to a quadratic minimization problem under linear equality constraints, we have to use the pseudo-inverse of a tensor product of matrices. However, one can prove that *the pseudo-inverse of a tensor product is the tensor product of pseudo-inverses*: this is a very useful property of tensor products since pseudo-inverses of linear operators pop-up in many applications, statistics and data analysis, for instance. This is what we shall do for our problem:

We posit the assumptions

$$\begin{cases} i) & c : X \to Y \;\&\; \psi : X \to Z_1 \text{ are continuous} \\ ii) & G : X \to \mathcal{L}(Z_2, Y) \text{ is continuous} \\ iii) & G(x) \in \mathcal{L}(Z_2, Y) \text{ is surjective for all } x \in X \end{cases}$$

and

the elements $\psi(a_p)$ are mutually orthonormal

Then the "heavy algorithm" associates with the synaptic matrix W_{P-1} the new synaptic matrix W_P defined by formula:

$$W_P \;=\; W_{P-1} - \psi(a_P) \otimes G(a_P)^{\ominus 1} \left(\Phi(a_P, W_{P-1}) - b_P\right)$$

This is also a Hebbian rule, in the sense that the correction of the synaptic matrix W_{P-1} is obtained by adding to it a matrix $p \otimes u$ whose entries $p_i e^j$ are proportional to the product of activities in the presynaptic and postsynaptic neurons. It uses only the former synaptic matrix W_{P-1} and the last pattern (a_P, b_P) for making the correction.

6 Newton algorithms

We recall that in the last analysis, a neural network with one layer, or several layers, or a continuous set of layers maps inputs x to outputs $y = \Phi(x, W)$.

We have to find synaptic matrices $\overline{W}$ learning one pattern, i.e., satisfying the equation

$$\Psi(W) \;:=\; \Phi(a, W) - b \;=\; 0$$

Instead of replacing these nonlinear problems by optimization problems to which we apply gradient methods, we may investigate the use of Newton algorithms:

$$\Psi'(W^n)\left(W^{n+1} - W^n\right) \;=\; -\delta\Psi(W^n) \;\text{ for all }\; n \geq 0$$

starting at some initial matrix W^0, where $\delta > 0$ is given. Since the linear operators $\Psi'(W^n)$ are not necessarily injective, we shall use the *slow* Newton algorithm, i.e., choose W^{n+1} such that the norm of the *discrete velocity* $W^{n+1} - W^n$ is minimal among the solutions of the above equation. By using the pseudo-inverse of the operators $\Psi'(E)$, the *slow* Newton algorithm can be written:

$$W^{n+1} - W^n \;=\; -\delta\Psi'(W^n)^{\ominus 1}\Psi(W^n) \;\text{ for all }\; n \geq 0$$

In the case of a one-layer network associated with a differentiable propagation function g, we have to solve the equation

$$\Psi(W) \;:=\; g(Wa) - b \;=\; 0$$

The slow Newton algorithm is

$$W^{n+1} \;=\; W^n + \delta a \otimes g'(W^n a)^{\ominus 1}\left(\frac{b - g(W^n a)}{\|a\|^2}\right)$$

The situation becomes more complicated for multi-layer networks. We are looking for a sequence of synaptic matrices learning a pattern $(a, b) \in X_0 \times X_L$, i.e., a solution $(W_1, \ldots, W_L)$ to the nonlinear equation

$$\Psi(W_1, \ldots, W_L) \;=\; \Phi_L(a, W_1, \ldots, W_L) - b \;=\; 0$$

We have to compute the differential $\Phi'_L(a, W_1^n, \ldots, W_L^n)$ to make explicit the slow Newton algorithm

$$\vec{W}^{n+1} - \vec{W}^n = \delta\,(\Phi'_L(a, W_1^n, \ldots, W_L^n))^{\ominus 1}\,(b - \Phi_L(a, W_1^n, \ldots, W_L^n)) \quad \text{for all } n \geq 0$$

starting at some initial matrix $\vec{W}^0$, where $\delta > 0$ is given.

If $(W_1^n, \ldots, W_L^n$ is a given sequence of synaptic matrices, we denote by x_l^n the solutions to the problem

$$x_l^n = g_l(W_l^n x_{l-1}^n) \quad \text{for all } l = 1, \ldots L$$

starting from $x_0 := a$.

We set $A_l^n := g'_l(x_l^n) W_l^n$ and

$$\begin{cases} G^n(l,k) &= g'_l(x_l^n) W_l^n \cdots g'_l(x_{k+1}^n) W_{k+1}^n \\ G^n(l,k)^\star &= W_{k+1}^{n\star} g'_{k+1}(x_{k+1}^n)^\star \cdots W_l^{n\star} g'_l(x_l^n)^\star \end{cases}$$

Let us assume that the propagation maps g_l are differentiable and that the derivatives g'_l satisfy

$$\sum_{k=1}^{L} \|x_{k-1}^n\|^2 G^n(L,k) g'_k(x_k^n) \quad \text{are surjective for all } n \geq 0$$

We denote by $\pi_L^n \in Y := X_L$ the solution to the equation

$$\left(\sum_{k=1}^{L} G^n(L,k) g'_k(x_k^n) g'_k(x_k^n)^\star G^n(L,k)^\star \|x_{k-1}^n\|^2 \right) \pi_L^n = \delta\,(b - x_L^n)$$

The sequence $(W_1^n, \ldots, W_L^n)$ of synaptic matrices W_l^n determined by the slow Newton algorithm for mapping the input a to the output b is given by the formula

$$W_k^{n+1} - W_k^n = x_{k-1}^n \otimes g'_k(x_k^n)^\star G^n(L,k)^\star \pi_L^n \quad \text{for all } k = 1, \ldots, L$$

The Newton algorithm keeps a Hebbian character.

These formulas can be adapted to continuous-layer networks and/or to the case of learning several patterns.

7 Discussion

In classification problems, synaptic matrices which learn a given set of patterns are equilibria of a nonlinear problem, which are computed through gradient and Newton type algorithms.

However, by looking only at static or asymptotic problems (mapping inputs onto outputs, or finding equilibria and more generally, attractors), the evolution mechanism is neglected and the "real" time used to model the neural networks is often replaced by artificial times. For instance, the algorithmic time, describing the iterations of a given algorithm, is often interpreted as a learning rule, although it may not be related to the time involved in the modeling of the network. Or, in the case of multi-layer networks, the layers of the network represent a second time scale.

By choosing the route used by most neural networks, one may bypass the basic question: Why and How do synaptic matrices evolve? and the basic answer: To adapt to *viability constraints* through *learning laws* which are feedbacks of the neural network regarded as a control system, associating at each instant with any signal a synaptic matrix allowing the adaptation to viability constraints. Among the attempts to answer such questions, we refer to a dynamical point of view presented in (Aubin J.-P., 1994) and the papers of Nicolas Seube.

Indeed, two recent mathematical developments contributed to enrich available tools for studying the class of particular systems arising in neural networks and more generally, in biological, social and economic sciences. Set-valued maps arise naturally in these fields, and set-valued analysis, which deals with the extension of the differential and integral calculus to set-valued maps, became a "must" in those domains (see for instance (Aubin J.-P. & Frankowska H., 1990)). Viability theory attempts to model Darwinian evolution by encapsulating the concept of (contingent) chance by differential inclusions instead of stochastic differential equations and necessity by state (or "viability") constraints to which at least one evolution starting from any initial state must comply (see for instance (Aubin J.-P. & Frankowska H., 1990, Aubin J.-P. 1991, Frankowska H., 1995))

8 References

⋆ ARBIB M.A. (1987) BRAINS, MACHINES, AND MATHEMATICS, Springer-Verlag

⋆ AUBIN J.-P. (1991) VIABILITY THEORY, Birkhäuser

⋆ AUBIN J.-P. (1993) OPTIMA AND EQUILIBRIA Springer-Verlag

⋆ AUBIN J.-P. (1994) NEURAL NETWORKS AND QUALITATIVE PHYSICS: A VIABILITY APPROACH Cambridge University Press

⋆ AUBIN J.-P. & FRANKOWSKA H. (1990) SET-VALUED ANALYSIS, Birkhäuser

AUBIN J.-P. & SEUBE N. (1991) *Apprentissage adaptatif de lois de rétroaction de systèmes contrôlés par réseaux de neurones*, C. R. Acad. Sci., Paris, 314, 957-963

AUBIN J.-P. & SEUBE N. (1993) *Apprentissage des feedbacks contrôlant des systèmes sous contraintes d'état par des réseaux de neurones*, contrat # 911411 passé par la Direction Recherche Etudes et Techniques

⋆ FRANKOWSKA H. (1995) SET-VALUED ANALYSIS AND CONTROL THEORY, Birkhäuser, Boston, Basel, Berlin

⋆ KOHONEN T. (1984) SELF-ORGANIZATION AND ASSOCIATIVE MEMORY. Springer-Verlag, Series in Information Sciences, Vol. 8

⋆ McCLELLAND J. L. , RUMELHART D. E. & (Eds.) (1986) PARALLEL DISTRIBUTED PROCESSING (VOL.I). M.I.T. Press

SEUBE N. (1993) *A State Feedback Control Law Learning Theorem*, in NEURAL NETWORKS IN ROBOTICS, 175-191 Klüwer Academic Publisher