Stony Brook University

Global Reinforcement Training of CrossNets

A Dissertation Presented

by

Xiaolong Ma

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Physics

Stony Brook University

December 2007

Stony Brook University

The Graduate School

Xiaolong Ma

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Konstantin K. Likharev - Dissertation Advisor
Distinguished Professor, Department of Physics and Astronomy

Peter W. Stephens - Chairperson of Defense
Professor, Department of Physics and Astronomy

Michael M. Rijssenbeek
Professor, Department of Physics and Astronomy

Paul R. Adams
Professor, Department of Neurobiology and Behavior

This dissertation is accepted by the Graduate School.

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Global Reinforcement Training of CrossNets

by

Xiaolong Ma

Doctor of Philosophy

in

Physics

Stony Brook University

2007

Hybrid "CMOL" integrated circuits, incorporating advanced CMOS devices for neural cell bodies, nanowires as axons and dendrites, and latching switches as synapses, may be used for the hardware implementation of extremely dense (10^7 cells and 10^{12} synapses per cm^2) neuromorphic networks, operating up to 10^6 times faster than their biological prototypes. We are exploring several "CrossNet" architectures that accommodate the limitations imposed by CMOL hardware and should allow effective training of the networks without a direct external access to individual synapses. Our studies have show that CrossNets based on simple (two-terminal) crosspoint devices can work well in at least two modes: as Hopfield networks for associative memory and multilayer perceptrons for classification tasks.

For more intelligent tasks (such as robot motion control or complex games), which do not have "examples" for supervised learning, more advanced training methods such as the global reinforcement learning are necessary. For application of global reinforcement training algorithms to CrossNets, we have extended Williams's

REINFORCE learning principle to a more general framework and derived several learning rules that are more suitable for CrossNet hardware implementation. The results of numerical experiments have shown that these new learning rules can work well for both classification tasks and reinforcement tasks such as the cart-pole balancing control problem. Some limitations imposed by the CMOL hardware need to be carefully addressed for the the successful application of in situ reinforcement training to CrossNets.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my advisor, Prof. Konstantin K. Likharev. His profound knowledge and immense creativity have been the best resource for my graduate study at Stony Brook University. The highest level of professionalism that he demonstrated in his teaching and research will continue to serve as a model for me in my future development.

I benefited much from discussions with Jung Hoon Lee, Dmitri Strukov and Jingbin Li. I appreciate their generous share of insights.

Valuable discussions with P. Adams, D. Hammerstrom, and T. Sejnowski are gratefully acknowledged.

Chapter 1

Introduction

1.1 Introduction to Artificial Neural Networks

The human brain is superior to a digital computer at many tasks which are hard to define mathematically or algorithmically. A good example is image processing, such as recognition of objects. A one-year old baby is much better at performing theses tasks than the best supercomputer. Furthermore, the human brain can deal with fuzzy information, and it is robust, fault tolerant, easy to adapt to new situation. These features and capabilities of human brain are much desirable in artificial information processing systems; yet very less is understood about its functionality [1].

The brain is composed of about 10^{11} neurons, which are connected to each other through about 10^{15} synapses. As shown in Fig. 1.1, the main part of a neuron is the cell body or soma. A long fiber called axon (which branches to strands that reach many other neurons) carries the output of the neuron, while

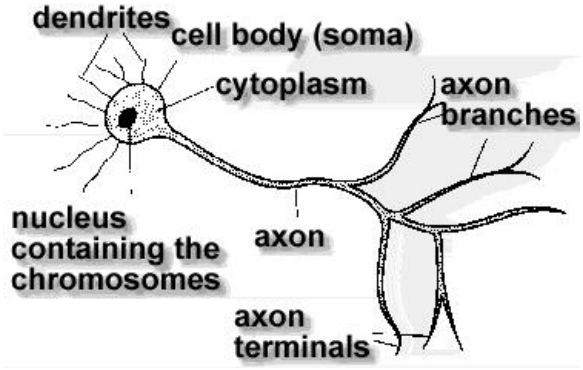


Figure 1.1: Schematics of a biological neuron.

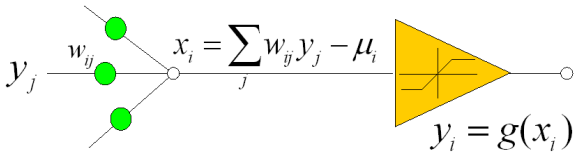


Figure 1.2: Schematics of the firing rate model of a neuron.

smaller fibers called dendrites receive the signals from other neurons through the coupling of synaptic junctions.

The process of transmitting signals form one cell to another is rather complicated. A simple model was proposed by McCulloch and Pitss [2] to describe a neuron as a binary threshold unit. It "fires" when the weighted sum of signals from other neurons exceeds certain threshold:

$$y_i(t+1) = \Theta(\sum_j w_{ij} y_j(t) - \mu_i), \tag{1.1}$$

where y_i is either 0 or one, and $\Theta(x)$ is the unit step function:

$$\Theta(x) \begin{cases} 1, & \text{if } x \leq 0; \\ 0, & \text{otherwise.} \end{cases} \tag{1.2}$$

Equation (1.1) can be generalized by replacing $\Theta(x)$ with a more general nonlinear function $g(x)$ (for example the tanh) called the "activation function". Without writing the time explicitly we denote the update rule as the following:

$$y_i = g(\sum_j w_{ij} y_j - \mu_i). \tag{1.3}$$

This model is schematically shown in Fig. 1.2

In Artificial Neural Networks (ANN), the neurons can be connected to each other in many different ways. In this work we discuss two most commonly seen architectures: the Hopfield Networks and the Multi-Layer Perceptrons (MLPs). The former is simply a fully connected, recurrent network in which every neuron is connected to all the others; the latter is a layered, feed-forward network shown in Fig. 1.3. [1]

Examples of applications of ANNs include associative memory (Hopfield Networks, [4]), speech generation (NETtalk [5]), face detection [6], handwrit-

[1] The input layer performs no computation and therefore sometimes not included in the layer count.

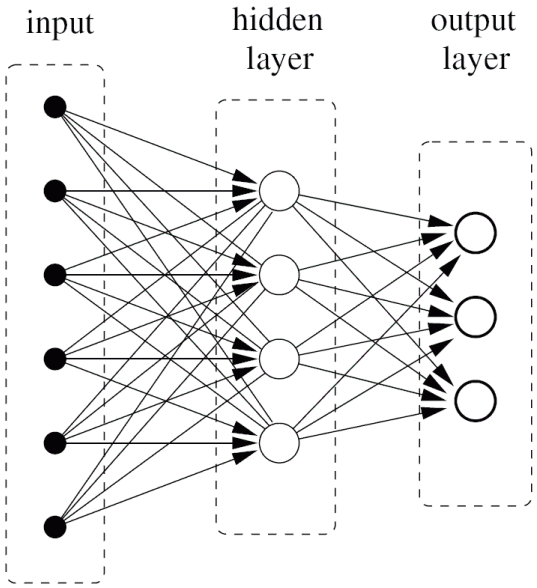


Figure 1.3: Three-layer perceptron. The cells are divided into layers; and there are only forward connections between adjacent layers (from Ref. [3]).

ten character recognition [7], and signal prediction and forecasting [1].

1.2 CMOL Circuits

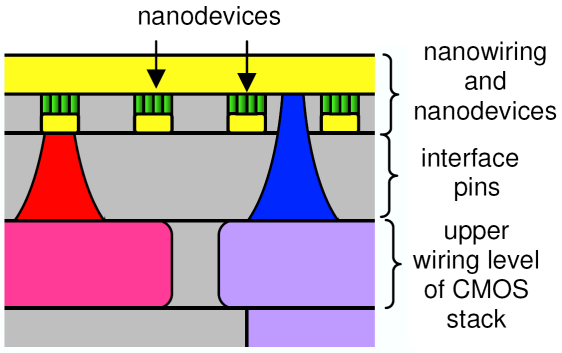
Recent results [8, 9], indicate that the current VLSI paradigm, based on a combination of lithographic patterning, CMOS circuits, and Boolean logic, can hardly be extended into a-few-nm region. The main reason is that at gate length below 10 nm, the sensitivity of parameters (most importantly, the gate voltage threshold) of silicon MOSFETs to inevitable fabrication spreads grows exponentially. As a result, the gate length should be controlled with a few-angstrom accuracy, far beyond even the long-term projections of the semiconductor industry [10]. (Similar problems are faced by the lithography-based single-electron devices [11].) Even if such accuracy could be technically implemented using sophisticated patterning technologies, this would send the fabrication facilities costs (growing exponentially even now) skyrocketing, and lead to the end of the Moore's Law some time during the next decade. This is why there is a rapidly growing consensus that the impending crisis of the microelectronics progress may be resolved only by a radical paradigm shift from the lithography-based fabrication to the "bottom-up" approach based on nanodevices with Nature-fixed size, e.g., specially designed molecules. Since

the functionality of such nanodevices is relatively low [9], they necessarily should be used as an add-on to a CMOS subsystem. Several proposals of such hybrid CMOS/nanodevice circuits were put forward recently (for their reviews, see Ref. [12–18]).The main idea of this combination is that the two-terminal devices have only one critical dimension (distance between two electrodes) which can be readily controlled, with sub-nanometer precision and without overly expensive equipment, by film thickness.

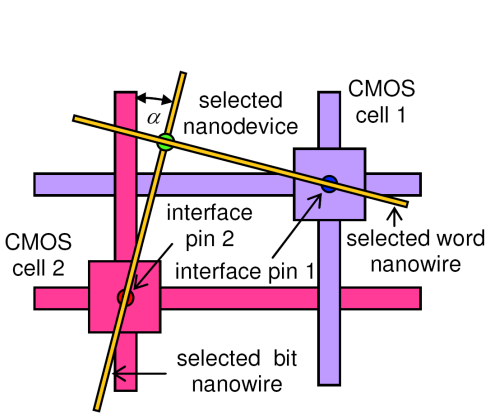
Our group is working on a particular circuit concept, dubbed CMOL [9, 16], for which the application prospects look best. As in several earlier proposals, nanodevices in CMOL circuits are formed (e.g., self-assembled) at each crosspoint of a "crossbar" array, consisting of two levels of nanowires (Fig. 1.4). However, in order to overcome the CMOS/nanodevice interface problems pertinent to earlier proposals, in CMOL circuits the interface is provided by pins that are distributed all over the circuit area, on the top of the CMOS stack. (Silicon-based technology necessary for fabrication of pins with nanometer-scale tips has been already developed in the context of field-emission arrays [19].) As Fig. 1.4(c) shows, pins of each type (reaching to either the lower or the upper nanowire level) are arranged into a square array with side $2\beta F_{\rm CMOS}$, where $F_{\rm CMOS}$ is the half-pitch of the CMOS subsystem, and β is a dimensionless factor larger than 1 that depends on the CMOS cell complexity. The nanowire crossbar is turned by angle $\alpha = \arcsin(F_{\rm nano}/\beta F_{\rm CMOS})$ relative to the CMOS pin array, where $F_{\rm nano}$ is the nanowiring half-pitch.

By activating two pairs of perpendicular CMOS lines, two pins (and two nanowires they contact) may be connected to CMOS data lines (Fig. 1.4(b)). As Fig. 1.4(c) illustrates, this approach allows a unique access to any nanodevice, even if $F_{\rm nano} \ll F_{\rm CMOS}$ - see Ref. [16] for a detailed discussion of this point. If the nanodevices have a sharp current threshold, like the usual diodes, such access allow to test each of them. Moreover, if the device may be switched between two internal states, e.g., as the single-electron latching switch [9, 20], each device may be switched into the desirable (ON or OFF) state by applying voltages $\pm V_{\rm W}$ to the selected nanowires, so that voltage $V = \pm 2V_{\rm W}$ applied to the selected nanodevice exceeds the corresponding switching threshold, while half-selected devices (with $V = \pm V_{\rm W}$) are not disturbed. (For details, see Sec. 1.3)

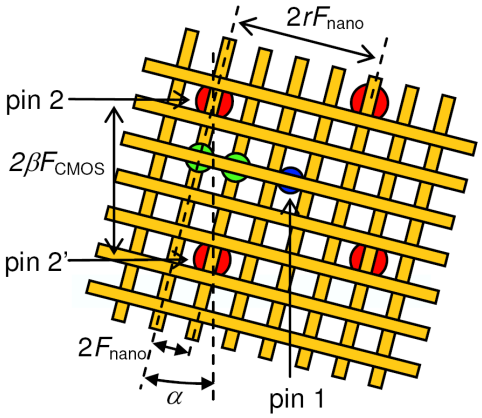
Two advantages of CMOL circuits over other crossbar-type hybrids look most important. First, due to the uniformity of the nanowiring/nanodevice levels of CMOL, they do not need to be precisely aligned with each other and the underlying CMOS stack (for details, see Ref. [3, 17, 18], thus allowing the use for nanowire formation of advanced patterning techniques [21–23] which lack precise alignment. This technique has already allowed to demonstrate



(a)



(b)



(c)

Figure 1.4: Low-level structure of the generic CMOL circuit: (a) schematic side view; (b) the idea of addressing a particular nanodevice, and (c) zoom-in on several adjacent interface pins to show that any nanodevice may be addressed via the appropriate pin pair (e.g, pins 1 and 2 for the leftmost of the two shown devices, and pins 1 and 2' for the rightmost device). On panel (b), only the activated CMOS lines and nanowires are shown, while panel (c) shows only two devices. (In reality, similar nanodevices are formed at all nanowire crosspoints.) Also disguised on panel (c) are CMOS cells and wiring.

crossbars with half-pitch F_{nano} of 17 nm [24] and 15 nm [25]. Second, recently there was a remarkable progress in fabrication of reproducible crosspoint devices with the necessary functionality of "latching switches" (see Sec. 1.3 for details), notably by Spansion LLC [26] and an IBM-led collaboration [27] - see Introduction to Ref. [28] for a review of recent literature. As a result, all major components of CMOL circuits may be considered demonstrated and ready for the beginning of a serious integration work [18]. Still, CMOL, similarly to all other nanodevice-based technologies, requires defect-tolerant circuit architectures, since the fabrication yield of such devices will hardly ever approach 100% as closely as that achieved for the semiconductor transistors (for example, see Sec. 2.3).

Application of CMOL circuits includes memories [29], boolean logic circuits [30] and neuromorphic networks [31]. The following chapters discuss CMOL application for neural networks.

1.3 CrossNets: CMOL Neuromorphic Networks

ANNs can be implemented on conventional computers by sequential algorithms. While this is adequate for many applications, only the hardware implementation can fully take advantage of the intrinsic parallelism of the neural network models [32]. There are numerous studies addressing the hardware implementation of neural networks through CMOS circuits. The latest reviews can be found in [33, 34]. Despite all the advances, the number of neurons and synapses in these implementations remain quite low compared to the human cerebral cortex [1].

Single electron devices can provide high densities and low power dissipation [35]. The idea that these devices can be used in neural networks was first proposed in Ref. [36] (and later developed in Ref. [37]), for a review, see Ref. [38]. None of these proposals come close to implementing a perceptron with adjustable weights.

We have proposed [39–45] a family of neuromorphic circuits, called Distributed Crossbar Networks ("CrossNets"), whose topology is uniquely suitable for CMOL implementation. Like most artificial neural networks explored earlier (see, e.g. References [1, 46–48]), each CrossNet consists of the following components:

1. Neural cell bodies ("somas") are relatively sparse and hence may be implemented in the CMOS subsystem. Most our results so far have been received within the simplest "firing rate" model, in which somas operate just as a differential amplifier with a nonlinear saturation ("activation") function (Fig. 1.5).

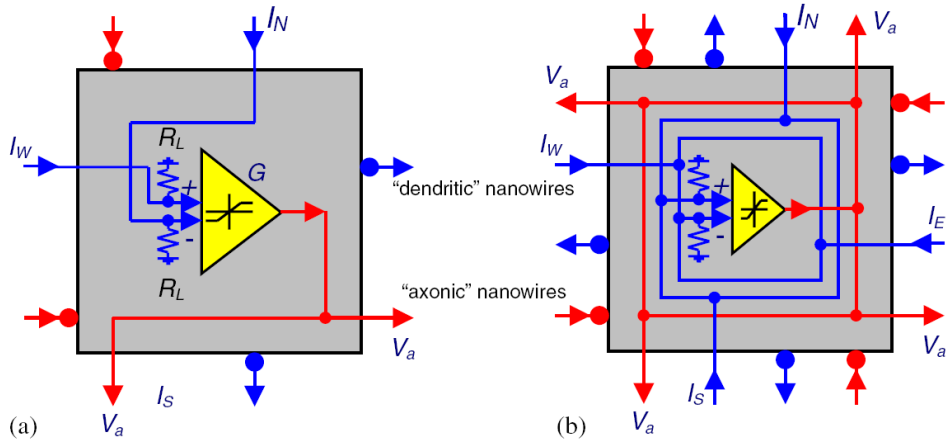


Figure 1.5: Structure of neural cell bodies (somas) of: (a) feedforward; and (b) recurrent CrossNets in the operation mode. Low input resistances R_L are used to keep all input ("dendritic") voltages $V_{\rm d} = R_L \sum_i I_i$ well below the output ("axonic") voltage $V_{\rm a}$, for any possible values of net input currents I_i , thus preventing undesirable anti-Hebbian effects [39]. G is the voltage gain of the somatic amplifier. Bold points show open-circuit terminations of nanowires, that do not allow somas to interact in bypass of synapses (see below).

2. "Axons" and "dendrites" are implemented as physically similar, straight segments of mutually perpendicular metallic nanowires (Fig. 1.4c). Somatic load resistances R_L (Fig. 1.5) keep all dendritic wire voltages $V_{\rm d}$ much lower than axonic voltages $V_{\rm a}$ (see Sec. 1.5.2 for an example of the effect of relatively large load resistance). Estimates show that wire resistances may be negligible in comparison with nanodevice resistances, even in the open state (see below). On the contrary, capacitance of the wires cannot be neglected and (in combination with R_L) determines the CrossNet operation speed.

3. Synapses, each comprising one or several similar nanodevices, are formed at crosspoints between axonic and dendritic nanowires (Fig. 1.4). Demonstrations of single-molecule single-electron devices by several groups can be found in Refs. [49–53], they provide an attractive option for synapse implementation. More recently, there has been a spectacular progress in the reproducible fabrication of the crosspoint devices based on metallic oxides, e.g., CuO_x [26], or doped GeSb [27], though scaling them below $\sim$ 10 nm may require using self-assembled monolayers (SAM) of specially designed molecules [18, 40, 45].

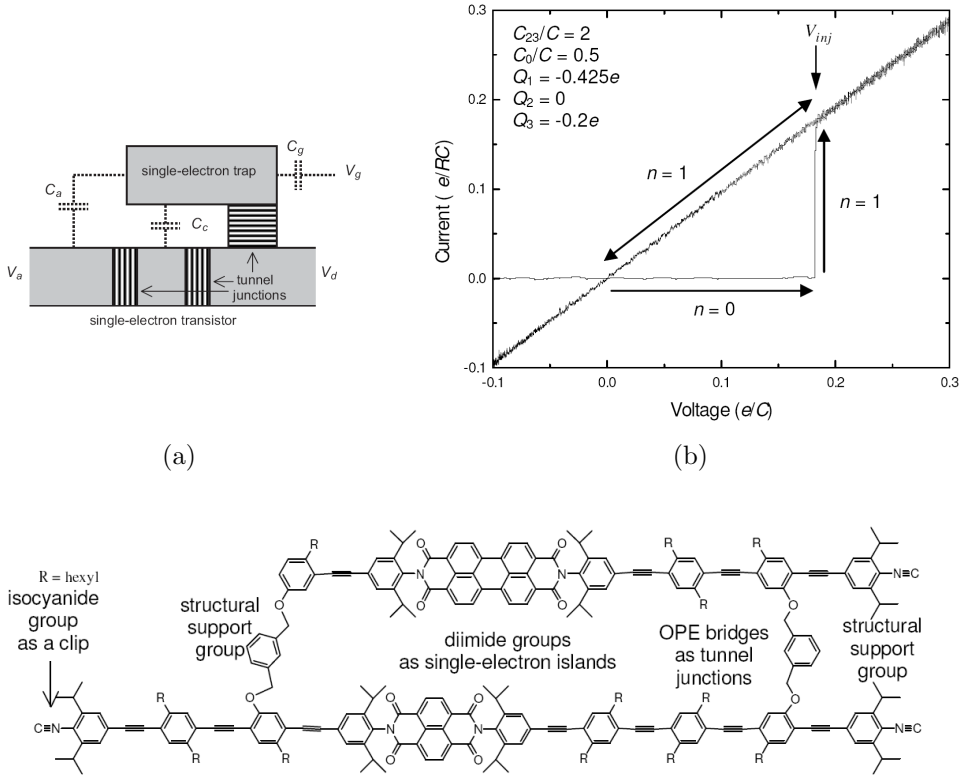


Figure 1.6: (a) Schematics; (b) functionality; and (c)possible molecular implementation of a two-terminal single-electron latching switch. The tunnel barrier connecting the box island is substantially thicker than those embedding the transistor island, so that the rate of tunnelling to and from the box is much lower. V_g is the voltage applied to the (quasi-) global gate (Fig 1.4(a)). Fig. 1.6(c) is courtesy of Prof. A. Mayr (SBU/Chemistry).

In this dissertation, we use the single-electron implementation as an example to illustrate the functionality of these two terminal crosspoint nanodevices.

Fig. 1.6(a) shows the schematics of the simplest single-electron device, latching switch, that has functionality (Fig. 1.6(b)) sufficient for CrossNet operation and allows a natural molecular implementation (Fig. 1.6(c)). The device is essentially a combination of two well-known single-electron devices: the transistor and the "trap" [11, 20, 54]. If the applied voltage $V = V_a - V_d$ is low, the trap island in equilibrium has no extra electrons ($n = 0$), and its total electric charge $Q = -ne$ is zero. As a result, the transistor is in the closed ("Coulomb-blockade") state, and source and drain are essentially disconnected. If V is increased beyond a certain threshold value V_{inj}, its electrostatic effect on the trap island potential (via capacitance C_s) leads to tunneling of an additional electron into the trap island: $n \to 1$. This change of trap charge affects, through the coupling capacitance C_c, the potential of the transistor island, and suppresses the Coulomb blockade threshold to a value well below V_{inj}; as a result, the transistor, whose tunnel barriers should be thinner than that of the trap, is turned into ON state in which the device connects the source and drain with a finite resistance R_0. (For a symmetric transistor, R_0 is close to the tunnel resistance of a single tunnel junction of the transistor [11].) Thus, the trap island plays the role similar to that of the floating gate in the usual nonvolatile semiconductor memories [55]. If V stays above V_{inj} , this connected state is sustained indefinitely; however, if the synaptic activity $V(t)$ remains low for a long time, the thermal fluctuations will eventually kick the trapped electron out, and the transistor will get closed, disconnecting the wires. This ON $\to$ OFF switching may be forced to happen much faster by making the applied voltage V sufficiently negative. [2] Thus the device works as an adaptive binary-weight, analog-signal synapse.

Fig. 1.7 shows the general topology of CMOL CrossNets on the examples of the simplest feedforward (a, c) and recurrent (b, d) networks. Red lines show "axonic", and blue lines "dendritic" nanowires. Dark-gray squares are interfaces between nanowires and CMOS-based cell bodies (somas), while light-gray squares in panel (a) show the somatic cells as a whole. (For the sake of clarity, the latter areas are not shown in the following panels and figures.) Note that the real area of the somatic CMOS cell may be much larger than that of the interface pin area of that cell; the former area is only limited by the distance between the adjacent somas. Signs show the somatic amplifier input polarities. Green circles denote nanodevices (latching switches) forming

[2] A virtually similar functionality may be achieved using configurational changes of specially selected molecules [56–58], however, such molecules are rather complex, and their switching may be too slow for most applications.

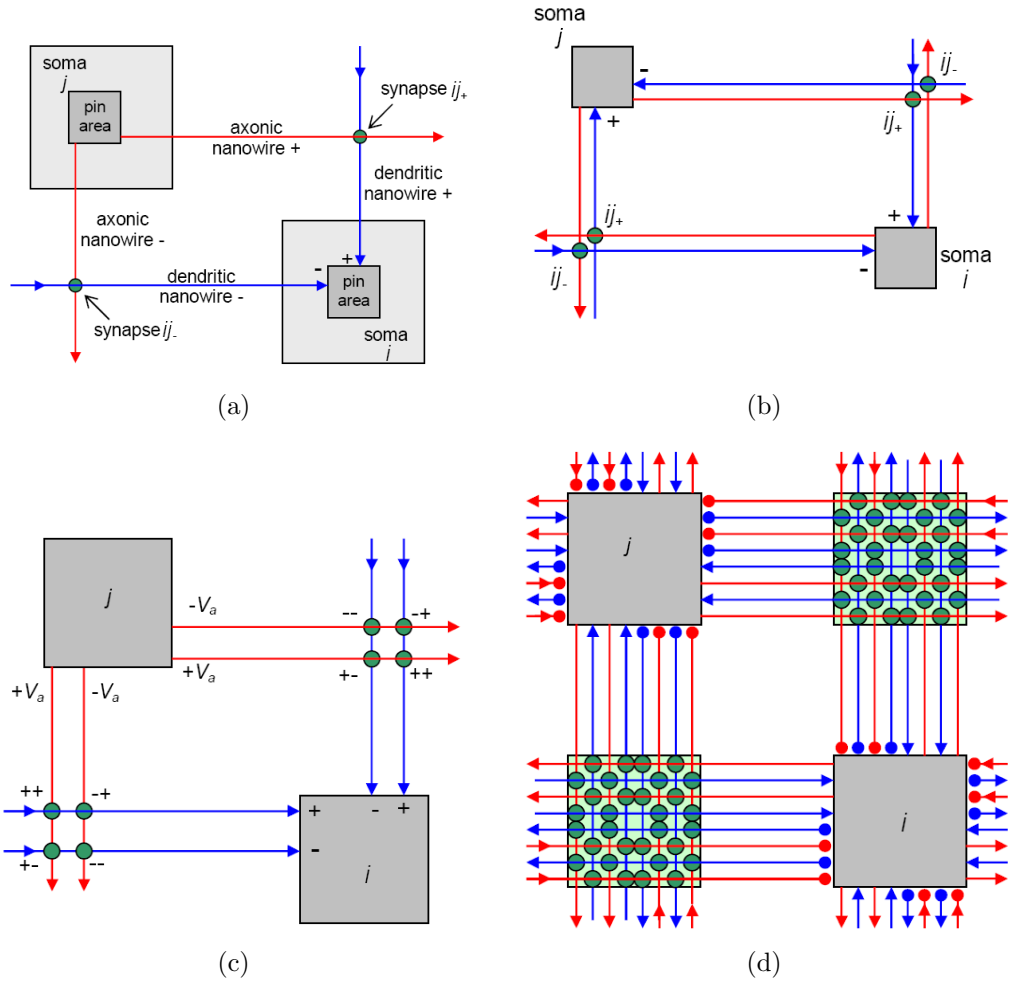


Figure 1.7: Schemes of cell connections in CrossNets: (a) simple (non-Hebbian) feedforward network, (b) simple recurrent network, (c) Hebbian feedforward CrossNet and (d) Hebbian recurrent CrossNet [44].

elementary synapses.

In Fig. 1.7(b), any pair of cells may be connected, in one way (from j to i, for example), by maximum two synapses leading to different somatic amplifier inputs, so that the net synaptic weight w_{ij} may take any of three values. (They may be normalized to -1, 0, and +1).

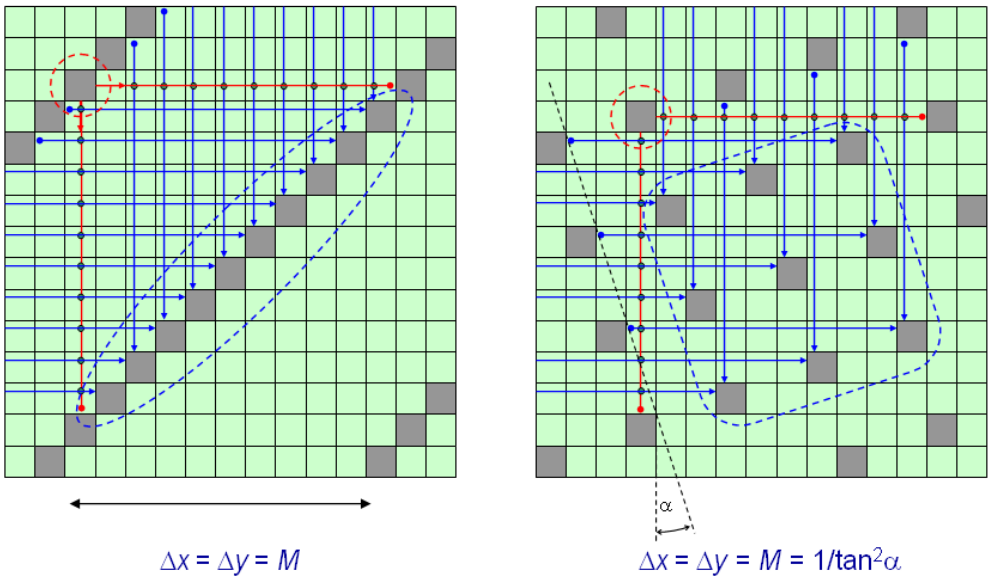
The nanowires are doubled in Fig. 1.7(c) to form the "Hebbian synapsis". Because both axonic and dentritic wires are doubled, this leads to 8 switches for one way connection. A group of four switches in different polarities ($++$, $+-$, $-+$ and $--$), either in the vertical or horizontal direction (the duplication is for the purpose symmetry), enables us to achieve the so-called Hebbian adaptation. (Please see Sec. 1.5.2 for details.)

For clarity the panels (a)-(c) show only the synapses and nanowires connecting one couple of cells (j and i). In contrast, Fig. 1.7(d) shows not only those synapses, but also all other functioning synapses located in the same "synaptic plaquettes" (painted light-green) and the corresponding nanowires, even if they connect other cells. (In CMOL circuits, molecular latching switches are also located at all axon/axon and dendrite/dendrite crosspoints; however, they do not affect the network dynamics, resulting only in approximately 50% increase of power dissipation.) The solid dots on panel (d) show open-circuit terminations of synaptic and axonic nanowires, that do not allow direct connections of the somas, in bypass of synapses.

CrossNet species differ by the number and direction of intercell couplings (Fig. 1.7) and by the location of somatic cells on the axon/dendrite/synapse field (Fig. 1.8). The cell distribution pattern determines the character of cell coupling. For example, the simplest CrossNet, the so-called FlossBar (Fig. 1.8(a)), in its feedforward version is essentially a Flavor of MLPs (Fig. 1.3) [1, 46–48], with quasi-local connectivity. Thus, the study of FlossBars allows a natural comparison of CrossNets with traditional artificial neural networks (typically implemented in software running on usual digital computers). In the so-called Inclined Crossbar (or just "InBar", see Fig. 1.8(b)), somatic cell pin areas are located on a square lattice that is inclined by a (small) angle α relatively to the axonic/dendritic nanowire array.[3] This geometry is more natural for CMOL implementation, because each somatic cell may have the same shape.

Also important is the average distance M between the somas, that determines connectivity of the networks, i.e. the average number of other cells coupled directly (i.e., via one synapse) to a given soma. The mechanism of this limitation is shown in Fig. 1.5(b) and 1.7(d): any axon running into a

[3]There is a substantial parallel between the incline angles α shown in Fig. 1.4(b) and 1.8(b). This analogy makes InBar arrays especially natural for CMOL implementation.



(a) (b)

Figure 1.8: Two particular CrossNet species: (a) FlossBar; and (b) InBar. For clarity, the figures show only the axons, dendrites, and synapses providing connections between one soma (indicated by the dashed circle) and its recipients (inside the dashed oval), for the feedforward case. For FlossBars, the number M of direct recipients is always even (in (a), equal to 10), while for InBars M is always the square of an integer number $M^{1/2} = 1/\tan\alpha$, where α is the angle of incline of the square lattice of somatic cells relative to the nanowire arrays. (In (b), $M = 9$.) In recurrent CrossNets (Fig. 1.5(b)), the cell connectivity is four times higher (equals $4M$).

somatic cell is open-circuit terminated (bold red points); so is any dendritic wire starting at a somatic cell (bold blue points). These terminations do not allow cell connections bypassing synapses, and set finite lengths for axons and dendrites and hence a finite connectivity: the farther are the somatic cells, the longer are the nanowire segments, and the more synapses they contact, providing connections to more cells. The most remarkable property of CMOL CrossNets is that the connectivity of these (quasi-)2D structures may be very large, for example, as high as 104, the number typical for the biological cortical networks with their quasi-3D structure [54, 59]). This property is very important for advanced information processing, and distinguishes CrossNets favorably from the so-called cellular automata with small (next-neighbor) connectivity which severely limits their functionality. (The price for the increase of connectivity is the operation speed-to-power tradeoff and noise immunity - see Sec. 1.6 below.)

While the somatic cell density in CrossNets is very important since it determines the network connectivity, the particular location of the cells is not too crucial (say, may be completely random [39, 42]) and may be directed by the convenience of either hardware implementation, or training, or both.

1.4 CrossNets: Modelling

Somatic cells were modelled by simple differential amplifiers (Fig. 1.5), with voltage gain G for small signals and sharp saturation at output levels $\pm V_0$:

$$V_a/V_0 = f[G(V_d^+ - V_d^-)/V_0] \tag{1.4}$$

Here V_d^+ and V_d^- are input voltage signals on, respectively, vertical and horizontal dendritic wires, $f(x)$ is the activation function which limits the range of the input signal. For example it can be the segmentally linear function

$$f(x) = \begin{cases} x, & \text{for } |x| < 1; \\ \text{sign}x, & \text{for } |x| > 1. \end{cases}, \tag{1.5}$$

or more frequently, we use the differentiable function $f(x) = \tanh(x)$ in our simulations.

Connected synapses are modelled by constant (Ohmic) resistances R_0 between the corresponding axonic and dendritic wires, disconnected synapses are treated as infinite resistances, while wire resistances are neglected. On the contrary, capacitance of dendritic wires is very important. In fact, for small load resistance R_L, the law of recharging of the dendrites by synaptic and load currents yields the system of differential equations describing the system

dynamics for soma i in Fig. 1.7(b):

$$C_0 M_i^{\pm} \frac{d(V_{\rm d}^{\pm})_i}{dt} = \sum_j w_{ij}^{\pm} \frac{(V_{\rm a})_j - (V_{\rm d}^{\pm})_i}{R_0} - \frac{(V_{\rm d}^{\pm})_i}{R_L} + I_i(t),$$
$$= \frac{V_0}{R_0} \sum_j w_{ij}^{\pm} f\left[\frac{G(V_{\rm d}^{+} - V_{\rm d}^{-})_j}{V_0}\right] - \frac{(V_{\rm d}^{\pm})_i}{R_i} + I_i(t), \qquad (1.6)$$

where

$$1/R_i = 1/R_L + \sum_j w_{ij}/R_0. \qquad (1.7)$$

In Eq. 1.6 C_0 is the dendrite capacitance per unit plaquette, $M_i^{\pm}$ is the length (measured as number of plaquettes) of horizontal/vertical dentritic wire, $I_i(t)$ is the external input current, and $w_{ij}^{\pm}$ is the net synaptic weight of the latching switch connecting axons of jth cell with ith dendrite, which may take binary values: 0 and 1. Each switch in Fig. 1.7 can be replaced with an array of n^2 switches and therefore implementing quasi-continuous synaptic weights (Fig. 1.12).

Subtracting the equation for $V_{\rm d}^{-}$ from that of $V_{\rm d}^{+}$, and denoting the normalized dentritic signal as

$$x_i(t) = G(V_{\rm d}^{+} - V_{\rm d}^{-})_i/V_0, \qquad (1.8)$$

we arrive at the following equations:

$$\tau_i dx_i/dt + x_i = g_i \sum_j w_{ij} f(x_j) + s_i, \qquad (1.9)$$

where $w_{ij} = w_{ij}^{+} - w_{ij}^{-}$ (for binary switches, $w_{ij} = -1, 0, 1$, $s_i(t) = I_i(t) G R_L/V_0$, $\tau_i = (M_i^{+} + M_i^{-}) R_i C_0$ and $g_i = G R_i/R_0$. Due to the isotropy of the Inbar structure, we have $M_i^{\pm} = M$ for all i; also, in the limit of $M R_L/R_0 \ll 1$, $R_i \approx R_L$, therefore the global time constant

$$\tau = 2 M R_L C_0, \qquad (1.10)$$

and the global effective voltage gain of the somatic cell

$$g = G R_L/R_0, \qquad (1.11)$$

and Eq. 1.9 becomes

$$\tau dx_i/dt + x_i = g\sum_j w_{ij} f(x_j) + s_i, \tag{1.12}$$

For hardware simulation, however, we usually use Eq. (1.6) in the following form (for the isotropic structures):

$$\Delta(V_{\rm d}^{\pm})_i = \frac{\Delta t}{R_0 C_0}\left(\sum_j w_{ij}^{\pm}[(V_{\rm a})_j - (V_{\rm d}^{\pm})_i] - (V_{\rm d}^{\pm})_i/r + I_i(t)R_0C_0\right)/M, \tag{1.13}$$

where $r = R_L/R_0$. In this case it is more convenient to measure time in R_0C_0. And if the voltages are measured in V_0, then Eq. (1.13) becomes unitless. In later demonstrations, we often show time in R_0C_0s. Although the time constant of the system is τ (Eq. 1.10), the limit of the speed of the system is ultimately determined by R_0C_0 (see Sec. 1.6 for details).

1.4.1 Self-excitation of Recurrent Networks

As an example, let us study the behavior of a recurrent InBar network (Fig. 1.8(b)) under Eqs. (1.12) without external inputs ($s_i = 0$ for all i). Suppose the synaptic weights are randomly initialized with a global probability p for the ON state for all the switches in the network, the typical behavior of the system is a chaotic oscillation [39], as shown in Fig. 1.9. In the experiments, we have used the four-group Hebbian synapses as shown in Fig. 1.7(c), but for simplicity, only the axonic wires are doubled (resulting 4 switches for one way connection).[4] If each switch is replaced with an array of n^2 switches (as shown in Fig. 1.12), the quasi-continuous weights $-2n^2 \le w_{ij} \le 2n^2$. The capacity of the dentritic wires per synaptic plaquette $C_0 \propto n^2$. To keep the time constant of the system unchanged, we need to reduce load resistance accordingly: $R_L \propto 1/n^2$ - see Eq. (1.10). We have also scaled the effective gain $g = 1/n$ to keep the amplitude of the dentritic signals constant. As we can see from Fig. 1.9, both the frequencies and amplitudes of the normalized dentritic signals (Eq. 1.8) becomes invariant with respect to n as a result of this scaling of hardware parameters (see the following for detailed discussions).

[4] In hardware implementation we may wish to double both the axonic and dentritic wires to achieve symmetric layout, as shown exactly in Fig. 1.7(c), but the results discussed in this section (and later) is still valid after simple adjustments to the parameters.

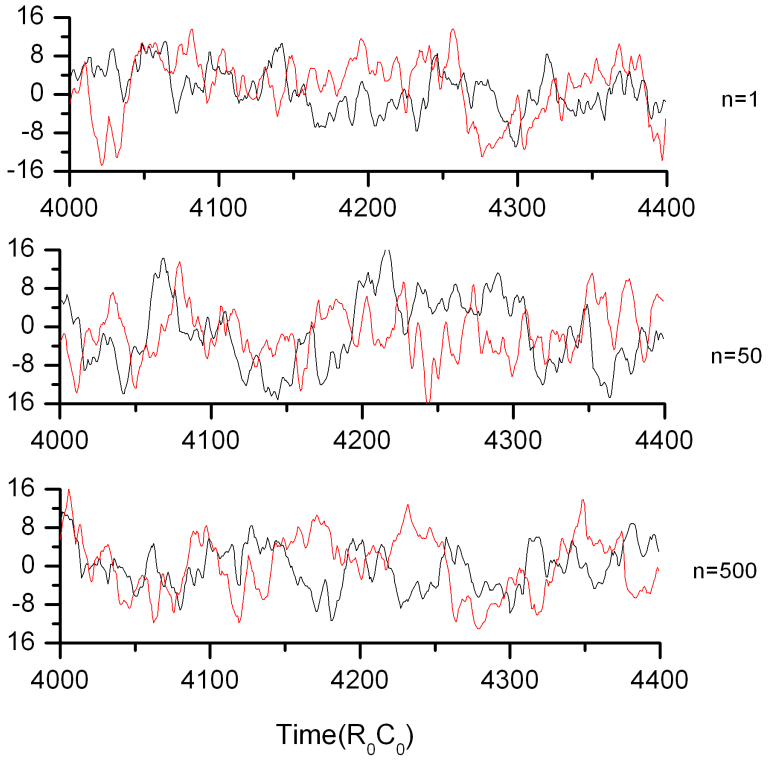


Figure 1.9: Sample normalized dentritic signals (x_i) from a recurrent Inbar with 425 neurons and connectivity $M = 16$. The system was simulated for $4 \times 10^4 \tau$ with a time step $\Delta t = 4\tau$. Other parameters are: $r = R_L/R_0 = 0.1/M/n^2$, and $g = 1/n$.

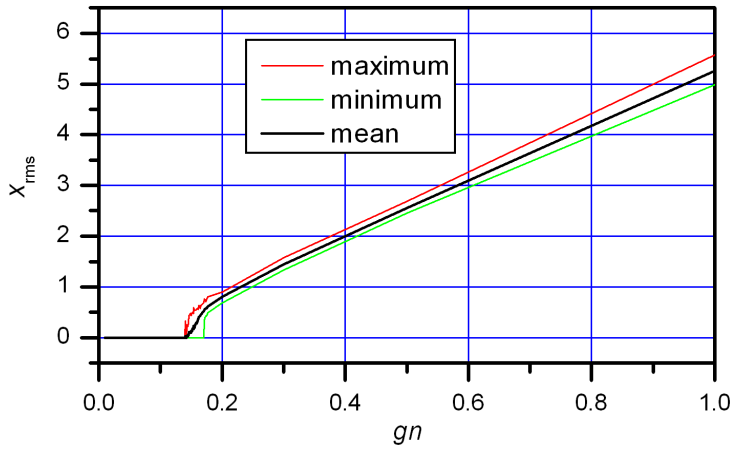


Figure 1.10: Self-excitation activity $x_{\rm rms}$ averaged over all neurons and the last $2 \times 10^4 \tau$. The system is the same as in Fig. 1.9. The black line is the mean values of 100 independent experiments (different initializations of synaptic weights) while the red and green ones are the maximum and minimum values, respectively.

The activity of the system over a time period from t_1 to t_2 measured by

$$x_{\rm rms} = \sqrt{\left\langle \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} x_i^2(t) dt \right\rangle}, \tag{1.14}$$

where the average $\langle \cdot \rangle$ is carried out over the statistical ensemble of all cells. $x_{\rm rms}$ is closely related to global parameters g and n. To see this, let us assume the activation function Eq. (1.5), and first study the case of very small gain. In this limit, because most signals x_i are small, we have $f(x) \approx x$, and Eqs. 1.12 become linear. Applying to these equations the Fourier transform ($X_i(\omega) = \mathcal{F}[x_i(t)] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-i\omega t} x_i(t) dt$), multiplying each equation by its complex conjugate, and averaging over the statistical ensemble of all cells, we get

$$\langle X_i X_i^* \rangle (1 + \omega^2 \tau^2) = g^2 \sum_{jj'} \left\langle w_{ij} w_{ij'} X_j X_{j'}^* \right\rangle, \tag{1.15}$$

where x^* is the complex conjugate of x. It is reasonable to assume that there is no statistical correlation between X_j and w_{ij}, because with large connectivity, each X_j depends many other synaptic weights collectively. Therefore,

$$\left\langle w_{ij} w_{ij'} X_j X_{j'}^* \right\rangle \approx \langle w_{ij} w_{ij'} \rangle \langle X_j X_{j'}^* \rangle. \tag{1.16}$$

Since each synaptic weight is a sum of four independent binomially distributed random numbers ranging from 0 to n^2, and the weights are independent with each other, we have $\langle w_{ij}w_{ij'}\rangle = 4n^2p(1-p)\delta_{j,j'}$. Because each cell is connected to 2M cells, the sum in Eq. (1.15) gives 2M equal terms, so that for variance $X^2 = \langle|X_i|^2\rangle$ we get the following equation:

$$X^2[1 + \omega^2\tau^2 - 8Mn^2p(1-p)g^2] = 0. \tag{1.17}$$

This equation gives us the critical value of g for the low-frequency excitation to arise ($\omega \approx 0$):

$$g_c = [8Mn^2p(1-p)]^{-1/2}. \tag{1.18}$$

In Fig. 1.9, $M = 16$, and $p = 0.5$, therefore $g_c n \approx 0.177$. This is confirmed by Fig. 1.10.

If the system is in the limit of deep saturation $g \gg g_c$, then most of the time $x_i \gg 1$, $f(x_i) = \text{sign}(x_i)$ and the amplitude of $f(x_i)$ no longer depends on x_i. Therefore Eq. (1.15) becomes

$$\begin{aligned}\langle X_iX_i^*\rangle(1+\omega^2\tau^2) &= g^2\sum_{jj'}\langle w_{ij}w_{ij'}\mathcal{F}[f(x_j)]\mathcal{F}[f(x_{j'})]^*\rangle,\\ &= g^2\sum_{jj'}\langle w_{ij}w_{ij'}\rangle\,\langle\mathcal{F}[f(x_j)]\mathcal{F}[f(x_{j'})]^*\rangle,\\ &= 8Mn^2p(1-p)g^2|F(\omega)|^2.\end{aligned} \tag{1.19}$$

where $F(\omega) = \langle|\mathcal{F}[f(x_j)]|^2\rangle$ does not depend on $X(\omega)$. This gives us[5]

$$X = \left[\frac{8Mn^2p(1-p)g^2|F(\omega)|^2}{1+\omega^2\tau^2}\right]^{1/2} \propto gn. \tag{1.20}$$

This relation can be confirmed by Fig. 1.9 and 1.10.

1.5 CrossNets: Training

In neural networks training means adjusting the strength of synaptic weights to achieve desired network behavior.

CrossNets can be trained as neural networks to perform information processing tasks, including the Hopfield type associative memories [39–44], pattern classification [61–63], and global reinforcement learning tasks [31, 64–66]. The

[5]According to Rayleigh's theorem [60], the same relation holds for the x_{rms} defined in Eq. 1.14

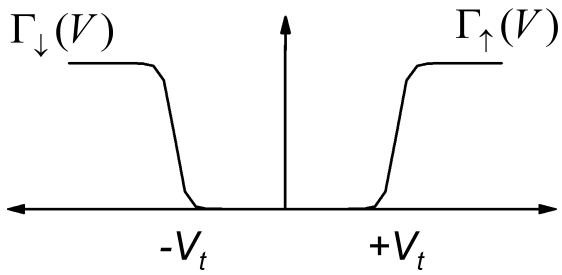


Figure 1.11: The model for the switching rates for the single-electron latching switches.

training of CrossNets are achieved through turning the switches (as shown in Fig. 1.6) ON or OFF by applying proper voltage $V = V_{\rm a} - V_{\rm d}$, and therefore adjusting the synaptic weights according to some training algorithms for neural networks. The ON and OFF switching processes (as all dynamics of single-electron devices) are random, the probability of having any latching switch in ON state is controlled by the following dynamics:

$$dp/dt = \Gamma_{\uparrow}(1-p) - \Gamma_{\downarrow}p, \tag{1.21}$$

where $\Gamma_{\uparrow}$ is the switching probability rate for turning the switch on if the switch is originally in the OFF state, and $\Gamma_{\downarrow}$ is the reverse. $\Gamma_{\uparrow\downarrow}$ are definite functions of the applied voltage V [9]. The model for these rates is schematically shown in Fig. 1.11. The switching rates are sharp functions of the applied voltage, growing rapidly from virtually zero to a high value at the corresponding voltage threshold, and saturating at certain level for large enough voltages. Below the saturating voltage, The "orthodox" theory of single-electron tunnelling predicts the rates to be exponentially-linear functions of V, close to the Arrhenius law

$$\Gamma_{\uparrow\downarrow} = \Gamma_0 \exp[\pm\beta(V+S)], \tag{1.22}$$

where $\beta \equiv e/k_{\rm B}T$, T is the effective temperature, while S is a shift parameter that depends on the switch design, and may be changed by applying voltage to a special global gate electrode (Fig. 1.6(a)). The desired average synaptic weight adjustment may be implemented using three different techniques, as discussed by the following sections.

1.5.1 Weight Import

For relatively small tasks, the training can be done outside and the resulted weights can be simply imported into the hardware. This procedure

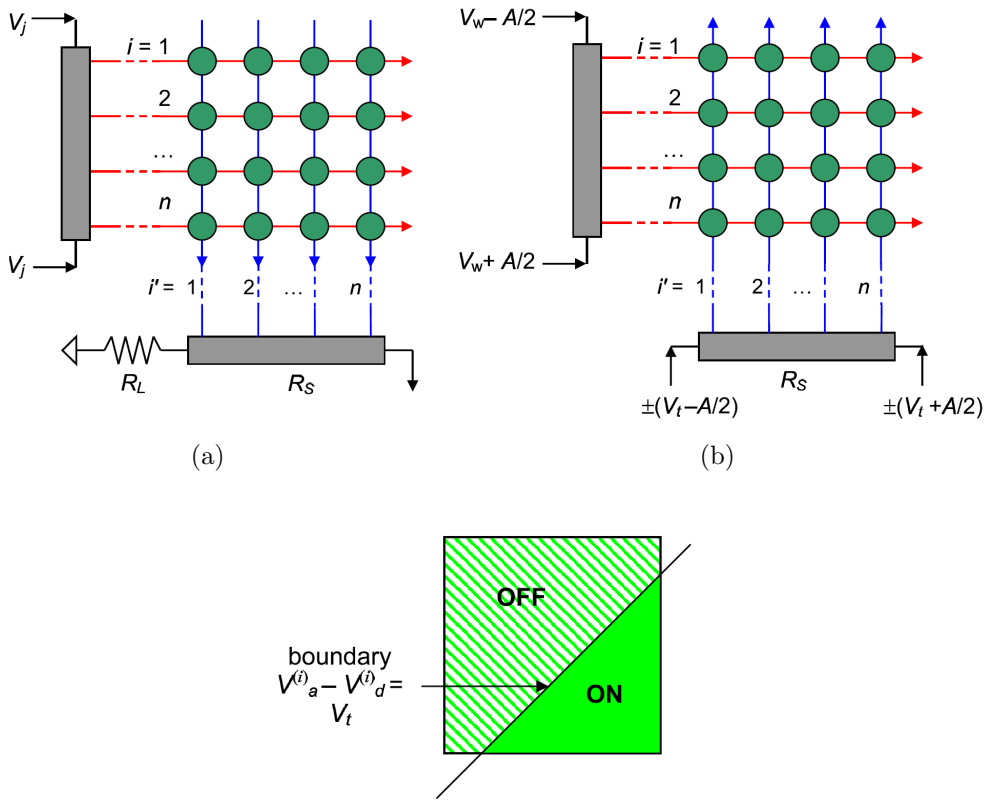


Figure 1.12: A half of the composite synapse for providing $L = 2n^2+1$ discrete levels of the weight in (a) operation and (b) weight import modes. The dark-grey rectangles are resistive metallic strips (with the total resistance $R_S \ll R_L$) serving as soma/nanowire interfaces. Plate (c) shows (schematically) the boundary between the domains of two possible states of elementary synapses.

is started with training of a homomorphic "precursor" artificial neural network with continuous synaptic weighs w_{ij}, implemented in software, using one of established methods (e.g., error backpropagation)[6]. Then the synaptic weights w_{ij} are transferred to the CrossNet, with some "clipping" (rounding) due to the discrete nature of elementary synaptic weights. To implement this transfer, all latching switches are first reset to their OFF state. For binary weights implemented as single switches, pairs of somatic cells are sequentially selected via CMOS level wiring. Using the flexibility of CMOS circuitry, these cells are reconfigured to apply external voltages $\pm V_W$ to the axonic and dendritic nanowires leading to a particular synapse, while all other nanowires are grounded. The voltage level V_W is selected so that it does not switch the synapses attached to only one of the selected nanowires, while voltage $2V_W$ applied to the synapse at the crosspoint of the selected wires is sufficient for its reliable switching.

In the composite synapses with quasi-continuous weights, only a part of the corresponding switches is turned ON or OFF. The details is discussed as follows. Suppose each switch in Fig. 1.7(b) is replaced by a square array of switches shown in Fig. 1.12, in the operation mode Fig. 1.12(a), all the n axons (red lines) are applied the same axonic voltage, while the currents in n dendrites are summed up, giving total weight $w = w^+ - w^-$, where w^+ and w^- is the number of switches in the ON state in each array. In order to fix the desirable value of weight w, one has to fix w^+ and w^-. To achieve this, during the training mode (Fig. 1.12(b)), somas apply graded voltages to their axons and dendrites:

$$V_{\rm a} = V_W + A(i/n - 1/2), \;\; V_{\rm d} = \pm[V_t + A(i'/n - 1/2)], \tag{1.23}$$

where $i (1 \le i \le n)$ is the nanowire number, the voltage spread A is slightly lower than V_t, and the sign of $V_{\rm d}$ is, as before, opposite for horizontal and vertical dentrites. This creates a gradient of the net voltage on the square grid of molecular devices; and in particular a boundary (Fig. 1.12(c)) that separates the region where the net voltage V applied to a molecule is less than the threshold voltage, V_t, from the region where $V > V_t$. This allows the neuron somas to adjust w^+ and w^-, and hence w. For further discussions, see Refs. [40, 44].

[6]In this case, better training accuracy may be achieved by using a discrete precursor that has the same quantization as in the hardware. For training of such software networks with discrete weights, please see Ref. [67].

1.5.2 Hebbian Adaptation

Let us consider a synapse that is composed of four switches with different polarity, as shown in Fig. 1.7(c) (but with the doubling of only the axonic wires):

$$w_{ij} = w_{ij}^{++} - w_{ij}^{+-} - w_{ij}^{-+} + w_{ij}^{--}. \tag{1.24}$$

According to Eq. (1.21) and Eq. (1.22), the dynamics of individual switches is

$$\begin{aligned} \frac{d < w_{ij}^{\pm\pm} >}{dt} &= \Gamma_0\{(1- < w_{ij}^{\pm\pm} >) \exp[\beta(V_{ij}^{\pm\pm} + S)] \\ &\quad - < w_{ij}^{\pm\pm} > \exp[-\beta(V_{ij}^{\pm\pm} + S)]\}, \end{aligned} \tag{1.25}$$

where $V_{ij}^{\pm\pm}$ are the voltages applied to each of the four groups. If the axonic voltages are transferred back to the dendrites for each cell, the voltage applied to the switches in the "Hebb training mode" is

$$V_{ij}^{\pm\pm} = (V_{\rm a})_j^{\pm} - (V_{\rm d})_i^{\pm} = (V_{\rm a})_j^{\pm} - (V_{\rm a})_i^{\pm}. \tag{1.26}$$

The somas are configured so that

$$(V_{\rm a})_i^{+} = -(V_{\rm a})_i^{-} = (V_{\rm a})_i. \tag{1.27}$$

By summing up the four equations for different polarities and transforming into the triangular form, the dynamics of the average synaptic weight $< w_{ij} >$ can be reduced to a simple formula if the synaptic weights are not close to saturation [40]

$$\frac{d < w_{ij} >}{dt} \approx -4\Gamma_0 \sinh[\beta S] \sinh[\beta (V_{\rm a})_i] \sinh[\beta (V_{\rm a})_j]. \tag{1.28}$$

Eq.(1.28) represents a correlation between the presynaptic and postsynaptic signals, which is exactly the fundamental idea of the well know Hebb-rule [1, 68] in neural computation. The Hebbian rule is a linear correlation between the pre- and post- synaptic signals:

$$\Delta w_{ij} = \eta V_i V_j. \tag{1.29}$$

where η is a constant learning rate. Although Eq. (1.28) is nonlinear, it prescribes the same "direction" for single weight change. We call the dynamics of four-group synapse in the training mode "Hebbian adaptation".

During operation, usually $V_{\rm d} \ll V_{\rm a}$. Therefore, if $V_{\rm a} < V_t$, then there is no adaptation in the "operation mode". However, we may use relative large

R_L such that $V_{\rm d}$ is comparable with $V_{\rm a}$, and hence achieve adaptation in the operation mode. In this case, the voltages applied to the switches are

$$V_{ij}^{\pm\pm} = (V_{\rm a})_j^{\pm} - (V_{\rm d})_i^{\pm}, \tag{1.30}$$

and generally

$$(V_{\rm d})_i^{+} \neq -(V_{\rm d})_i^{-}. \tag{1.31}$$

It is straightforward to prove that in the operation mode,

$$\begin{aligned} \frac{d < w_{ij} >}{dt} \approx\ & 4\Gamma_0 \sinh[\beta (V_{\rm a})_j] \sinh\left[\beta \frac{(V_{\rm d})_i^{+} - (V_{\rm d})_i^{-}}{2}\right] \\ & \sinh\left[\beta\left(\frac{(V_{\rm d})_i^{+} + (V_{\rm d})_i^{-}}{2} - S\right)\right]. \end{aligned} \tag{1.32}$$

Since $(V_{\rm d})_i^{+} - (V_{\rm d})_i^{-}$ always has the same sign as $(V_{\rm a})_i$, we can guarantee Hebbian or anti-Hebbian adaptation provided that the strength of the global shift is large enough

$$|S| > \left|\frac{(V_{\rm d})_i^{+} + (V_{\rm d})_i^{-}}{2}\right|, \text{ for all } j. \tag{1.33}$$

Hebbian adaptation, either achieved by Eq. (1.28), Eq. (1.32), or by the random multiplication technique discussed in the next section, is very important for virtually all of the training algorithms known for neural networks. When $\eta > 0$, the Hebb's rule Eq. (1.29) reinforce the current input-output relation by strengthening the correlation between pre- and post- synaptic signals. When $\eta < 0$ (in this case it is called anti-Hebbian rule), on the other hand, it has the opposite effect. This is useful in the global reinforcement training discussed in latter Chapters. For details, please see Sec. 4.1.

The following chapters will be focused on training CrossNets as Hopfield networks and reinforcement learning agents, both relies on the Hebb's rule for training. CrossNets can also be trained as pattern classifiers using similar Hebbian type of adaptation. For details, please see Refs. [61–63].

1.5.3 Time Multiplexing

For room temperature operation, the single-electron island size should be so small (below 1 nm) that electron motion in it is substantially quantum-confined, making the energy spectrum discrete [9]. Theory [69] shows that in this case, for a substantially large distance between the switching threshold values, $2V_t \gg k_{\rm B}T/e \approx 26$ meV, the dependence $\Gamma_{\uparrow\downarrow}(V)$ as shown Fig. 1.11

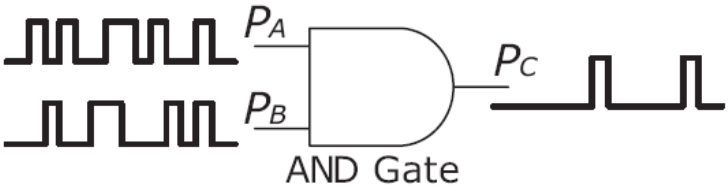


Figure 1.13: Stochastic multiplier using a single AND gate. P_A, P_B and P_C are firing probabilities of the input and output streams; and $P_C \approx P_A P_B$ (form Ref. [70]).

may be well approximated by step functions.[7]

Let us assume that below the threshold $\pm V_t$, the switching rates $\Gamma_{\uparrow\downarrow}$ are 0; while above the threshold they are constants independent of the magnitude of the voltages applied to the switches. The strength of the adaptation can be controlled instead by the time duration of the voltages. According to Eq. (1.21),

$$\Delta p = [\Gamma_\uparrow (1-p) - \Gamma_\downarrow p]\Delta t. \tag{1.34}$$

In this case, the multiplication of two signals (which is needed for Hebbian type of adaptation), can be achieved by "stochastic multiplication" [62]. The basic idea is to represent real valued signals using stochastic bit streams [71]. This allows a dramatic simplification of the circuitry required to implement many devices, since the multiplication of two values may be performed by a simple AND [70, 72] or XOR [73] gate. For example, see the illustration of Fig. 1.13. The following explains how this is achieved by time multiplexing in CrossNets.

As shown in Fig. 1.14, each synapse consists of four groups (arrays) of $n \times n$ elementary latching switches. In the operation mode, the open switches supply currents, proportional to axonic voltages, to the inputs of differential dendritic amplifiers, with different polarities. As a result, the net synaptic weight is

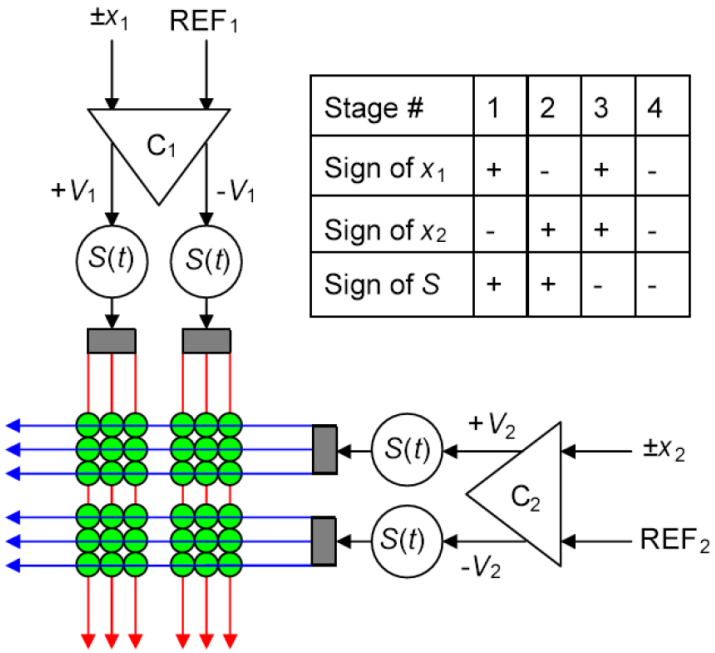
$$w = w_{\max}\frac{N}{2n^2} \tag{1.35}$$

$$N = N_{++} + N_{--} - N_{+-} - N_{-+} \tag{1.36}$$

where each $N_{pp'}$ is the number of ON-state switches in each group ($0 \le N \le n^2$), so that w can take $L = 4n^2 + 1$ quantized values.

In the training mode, signals that are to be multiplied ($x_{1,2} \in [0, x_{\max}]$) are compared with random signals $REF_{1,2}$, which are independent and evenly

[7] The same is true for other types of latching switches [17, 18, 63].



Stage #	1	2	3	4
Sign of x_1	+	-	+	-
Sign of x_2	-	+	+	-
Sign of S	+	+	-	-

Figure 1.14: The scheme for in-situ training. Reference signals $REF_{1,2}$ are random analog signals. Each small circle is the latching switch; the composite synapse consists of four groups of $n \times n$ similar switches. (The figure is for the case $n = 3$). $C_{1,2}$ are the signal comparators with binary output signals. The alternating global shift signal $S(t) = S_0$, together with flipping the signs of signals $x_{1,2}$, performs time division multiplexing (from Ref. [62]).

distributed within the same range $[0, x_{\max}]$. The comparator C_i generates voltage $V_i \approx \text{sign}(\pm x_i)(V_t/2)$, when $x_i > REF_i$, and 0 otherwise. This produces bit steams with densities of nonzero values proportional to x_i. The switches naturally perform the AND function because the net voltage can not exceed the threshold unless both V_1 and V_2 are nonzero (similar to the discussion in Sec 1.5.1). A global shift signal $S(t) = \pm S_0$, with an amplitude slightly below $V_t/2$, is added to the output voltages to further differentiate among the groups of switches and allow us to select only one of the four for update in each stage. The sign of x_i and $S(t)$ alternates according to the table to perform the 4-stage time division multiplexing.

The overall result is that the switches are updated with time/probability proportional to $x_1 x_2$:

$$\langle \Delta w \rangle \propto \Delta t x_1 x_2 \times \begin{cases} w_{\max} - w, & \text{when } x_1 x_2 > 0; \\ w_{\max} + w, & \text{when } x_1 x_2 < 0, \end{cases} \tag{1.37}$$

where the relaxation term w is due to the fact that only the switches that are in the off state can be turned on (for details see Ref. [62]).

1.6 Performance Estimation

We can estimate the time constant and power consumption of CrossNets. The most fundamental limitation on the half-pitch F_{nano} (see Fig. 1.4) comes from quantum-mechanical tunnelling between nanowires. If the wires are separated by vacuum, the corresponding specific leakage conductance becomes uncomfortably large ($10^{12}\Omega^{-1}\text{m}^{-1}$) only at $F_{\text{nano}} = 1.5\text{nm}$; however, since realistic insulation materials (SiO_2, etc.) provide somewhat lower tunnel barriers, let us use a more conservative value $F_{\text{nano}} = 3\text{nm}$. With the typical specific capacitance of $3 \times 10^{-10}\text{F/m} = 0.3\text{aF/nm}$ [40], this gives nanowire capacitance $C_0 \sim 1\text{aF}$ per working elementary synapse, because the corresponding segment has length F_{nano} (see Fig. 1.7(c)).

The CrossNet operation speed is determined by the time constant of dendrite nanowire capacitance discharging through the load resistance (see Fig. 1.5): $\tau = R_L C_0$ (see Eq. 1.10). This resistance can not be made arbitrarily small, however, because the effective gain $g \propto R_L/R_0$ (see Eq. 1.11), where R_0 is the resistance of a latching switch in the "on" state. To keep desired functionality, let us assume g is hold constant. Then the limit of speed actually comes from R_0, because the switch resistances are the main power consumers of the system.

Therefore, let us define $\tau_0 = R_0 C_0$ (which is also used as unit to measure

time in hardware simulations) as the measurement of the the speed the system. For example, the time of image recovery for Hopfield type operation (see Fig. 2.2) is approximately $6\tau_0$.

The possibilities of reduction of R_0, and hence τ_0, are limited mostly by acceptable power dissipation per unit area, that is close to $V_0^2/(2F_{\rm nano})^2 R_0$. For room temperature operation, the voltage scale $V_0 \approx V_t$ should be of the order of at least $30k_BT/e \approx 1$V to avoid thermally induced errors. With our number for $F_{\rm nano}$, and a relatively high but acceptable power consumption of 100W/cm^2, we get $R_0 \approx 10^{10}\Omega$ (which is a very realistic value for single molecule single-electron devices like one shown in Fig. 1.6(c)). With this number, τ_0 is as small as ~ 10ns[8] This means that the CrossNet speed may be at least six orders of magnitude higher than that of cerebral cortex circuitry. Even scaling R_0 up by a factor of 100 to bring power consumption to a more comfortable level of 1W/cm^2, would still leave us at least a four-orders-of-magnitude speed advantage.

[8]The real time constant τ is even r times smaller, where $r = R_L/R_0$ is typically much smaller than 1. The actually value of r depends on the effective gain g that the network is operating at, as well as the gain of the differential amplifier G.

Chapter 2

CMOL CrossNets Operating in Hopfield Mode

2.1 Introduction to Hopfield Networks

The Hopfield networks [4] are simple examples of how collective computation can work [1]. The task is to store P patterns into a fully connected neural network (for example, in a network with N cells, each cell in the network is connected to all the other $N-1$ cells) so that when the network is presented with a partial or damaged image, the original pattern can be reconstructed (Fig. 2.1).

We denote the ith pixel in the μth pattern by ξ_i^μ, where $\mu = 1, 2, ..., P$ and $i = 1, 2, ..., N$, and N is the size of both the patterns and the network. Let us assume black and white images and $\xi_i = \pm 1$. The solution to this problem is the so-called Hebb's prescription:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu. \tag{2.1}$$

The idea is based on Hebb's hypothesis [68] that the synaptic weight adaptation should be made according to the correlation between pre- and post-synaptic signals (see Eq. 1.29).

The Hopfield networks have a physical analogy to magnetic systems. An "energy function" can be defined for the network

$$H = -\frac{1}{2} \sum_{ij} w_{ij} y_i y_j, \tag{2.2}$$

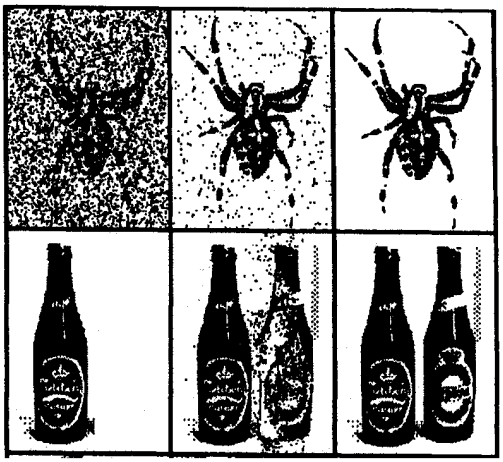


Figure 2.1: Example of associate memory. The middle column shows some intermediate states (from Ref. [1]).

where y_i are the outputs of the cells. It can be shown that for the weights defined by Eq. (2.1), the patterns to be stored are natural stable states (or attractors) for which the energy function has local minima. Actually the networks can be made stochastic [74, 75] to introduce the concept of "temperature"; and the Ising model [76] and the minefield theory in statistical physics can be applied to analyzing the Hopfield networks[77].

For a fully connected McCulloch-Pitss network (Eq. 1.1) of size N, the storage capacity, for 99% of correct recall for example, can be shown to be [1]:

$$P_{\max} = 0.15N \tag{2.3}$$

2.2 Hopfield Network in CrossNets

In a isotropic Inbar CrossNet (Fig. 1.8(b)), the time evolution of the signals in the circuits (without external inputs) are described by Eq. (1.12):

$$\tau dx_i/dt + x_i = g \sum_j w_{ij} f(x_j). \tag{2.4}$$

In the discussions in this chapter we use the segmentally linear activation function (Eq. 1.5):

$$f(x) = \{ \begin{array}{ll} x, & \text{for} |x| < 1 \\ \text{sign}x, & \text{for} |x| > 1 \end{array} . \tag{2.5}$$

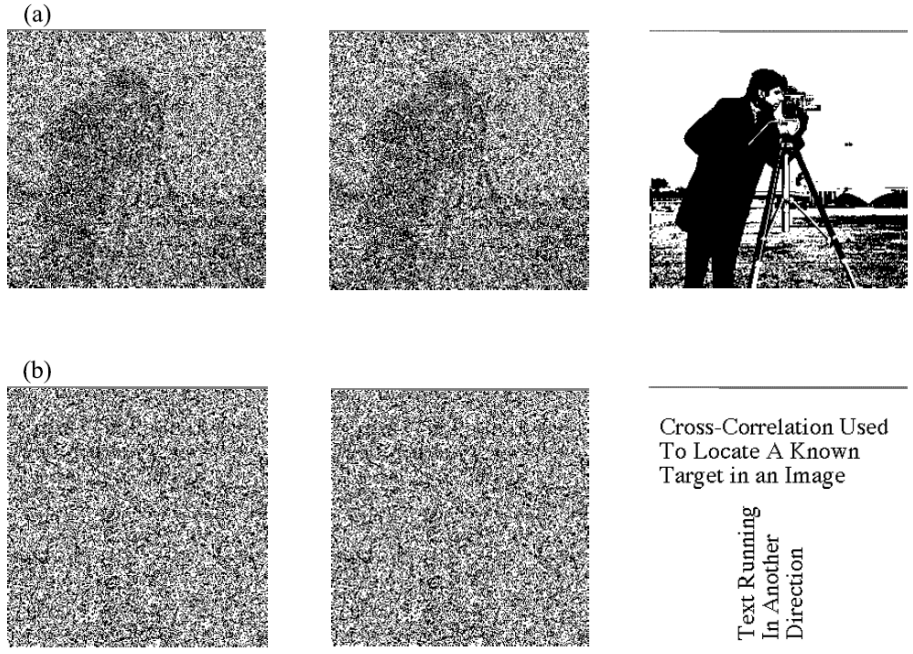


Figure 2.2: An InBar with $N = 256 \times 256$ neurons and $4M = 64$ connectivity in the Hopfield operation mode. The system is taught four images. Recall of two images are shown. After initialized to one of the patterns with 40% of bits flipped, the system converges to the original image with 100% fidelity in $\sim 6R_0C_0$ where R_0 is the resistance of a molecular device on the "on" state, and C_0 is the capacitance of nanowires per one plaquette. (From Ref. [3].)

For symmetric weights, i.e., $w_{ij} = w_{ji}$, Eq.(2.4) always reaches the following stable equilibrium solution [1]:

$$x_i = g \sum_j w_{ij} f(x_j). \tag{2.6}$$

To implement Hopfield network in CrossNets with binary weights, we set the synaptic weights according to the "clipped Hebb rule":

$$w_{ij} = \text{sign} \sum_{\mu=0}^{P-1} \xi_i^\mu \xi_j^\mu, \tag{2.7}$$

The exact solution of the "clipped" Hopfield Model based on Ising Model is given in [78]. Setting weights to externally calculated values can be readily achieved by the weight import procedure described in Sec. 1.5.1. An example of such implementation is shown in Fig. 2.2.

Unlike the traditional fully connected Hopfield model discussed in the previous section, the connectivity CrossNets is limited. For an recurrent Inbar CrossNet of N cells, each cell is connected to $4M$ neighbors(see Fig. 1.8(b)), where $4M < N$ is the connectivity of the network. The capacity of this network with localized connectivity can be estimated using similar method as in Ref. [1]. The maximum number of patterns the system can recall with 99% correct bits is (for details, see Ref. [39]):

$$P_{\max} \approx \alpha_2(4M), \tag{2.8}$$

with $\alpha_2 = 0.118$.

2.3 Defect Tolerance

The Synapse in the CrossNets can be implemented by self-assembling molecular latching switches described in [39]. The practical fabrication of those molecular devices with extremely high density will never achieve a 100% yield. Luckily, thanks to parallel processing of information, neural networks are naturally robust and fault tolerant [1]. This is one of the motivations behind neural network application of CMOL circuits. This section studies the effect of a small fraction of bad devices to the network and shows that defect tolerance of CrossNets running in Hopfield mode is exceptionally high. Detailed studies on defect tolerance of CrossNets operating as pattern classifiers, as well as training methods specifically designed to address this problem can

be found in Ref. [63].

We will use the same defect model which has been accepted in other studies of hybrid semiconductor/nanodevice circuits [28, 79–81], namely that the crosspoint latching switch defects are independent (with probability p) and are equivalent to "stuck-at-open" faults, i.e. that the bad device is either missing (leaving the nanowires disconnected) or is always in the OFF state (Fig. 1.6). It is believed that most of the defects in molecular devices will just cause latching switches to be disconnected. Therefore we simulate the defects by randomly setting synaptic weights w_{ij} to zero. If the fraction of bad devices is small, the dynamics of the system will not change much, and the result of defect is just a lower connectivity $M_{\text{effective}} < M$. Considering the fact that each pair of connected somatic cells are connected by two Synapse simultaneously (see Fig. 1.7(b) and Ref. [39]), and that we need to disconnect both synapses in order to fully cut the connection between two cells, we can obtain the formula for effective connectivity:

$$M_{\text{effective}} = M(1 - p^2). \tag{2.9}$$

Now we can calculate the capacity of the network with defects, following similar arguments as in [39].

Without losing generality, let us examine the stability of the first pattern ($\mu = 0$). To achieve consistence between the input and desired output, we require:

$$\text{sign}(x_i^0) = \xi_i^0, \tag{2.10}$$

for all i. According to Eq. (2.7),

$$x_i^0 = \sum_j w_{ij}\xi_j^0 = \sum_j \xi_j^0 \text{sign}\left(\sum_\mu \xi_i^\mu \xi_j^\mu\right), \tag{2.11}$$

if all outputs are clamped at the correct value ($f(x_j) = \xi_j^0$) in Eq. (2.6). (For simplicity we have set $g = 1$.) We define variable Z_i as the desired output of cell i, multiplied by it's input:

$$Z_i = \xi_i^0 x_i^0 = \sum_j \text{sign}\left(\xi_i^0 \xi_j^0 \sum_\mu \xi_i^\mu \xi_j^\mu\right) = \sum_j \text{sign}(1 + \delta_{ij}) \tag{2.12}$$

where we have moved $\xi_i^0 \xi_j^0$ into the signum and,

$$\delta_{ij} = \sum_{\mu \neq 0} \xi_i^0 \xi_j^0 \xi_i^\mu \xi_j^\mu \tag{2.13}$$

is the noise term. Because of defects in the network, we should keep in mind that the summation is only carried over js for which $w_{ij} \neq 0$, so there are only $4M_{\text{effective}}$ terms instead of $4M$ terms in the summation over signum. For large P and M, both δ_{ij} and Z_i are random walks with step size 1, and the distributions are approximately Gaussian. If we define the output of cell i to be "correct" when it produces the correct sign, i.e. $f(x_i)\xi_i^0 \geq 0$ or $Z_i \geq 0$, then we will get the same capacity as in Eq. (2.8), only with M replaced by $M_{\text{effective}}$:

$$P'_{\max} = \alpha_2(4M)(1-p^2). \tag{2.14}$$

Therefore, if the corruption rate of the switches is p, the capacity is decreased by a factor of p^2.

In our numerical simulation, however, the number of patterns is small(ranging from 3 to 8). Moreover, we had adopted a more strict yet more practical criteria for "correct" bits: the output of cell i is correct if $f(x_i) = \xi_i^0$. To fully explain the experiment data, we need more exact calculations than that in Ref. [39].

First of all, the noise distribution is no longer a Gaussian. We use the exact Binomial distribution (for equal probability of ± 1):

$$\begin{aligned} f_\delta(x) &= \text{Prob}[\delta_{ij} = x] \\ &= \text{Binomial}[P-1, \frac{x+P-1}{2}] = \frac{1}{2^n} C_{P-1}^{\frac{x+P-1}{2}}. \end{aligned} \tag{2.15}$$

For discrete distribution, one has to consider the situation when the signum is zero. The zero terms has no effect to the random walk and hence should be excluded from summation in Eq.(2.12) when one calculates the distribution. The actual number of terms in the summation over signum is therefore

$$n = 4M_{\text{effective}}(1-p_0), \tag{2.16}$$

where $p_0 = f_\delta(-1)$ is the probability for the signum to be zero. The probabilities of the signum being +1 and -1, respectively, are:

$$p_+ = \frac{1}{1-p_0} \sum_{x=0}^{P-1} f_\delta(x), \tag{2.17}$$

$$p_- = \frac{1}{1-p_0} \sum_{x=-P+1}^{-2} f_\delta(x), \tag{2.18}$$

where the factor $1/(1-p_0)$ is to normalize p_+ and p_- so that $p_+ + p_- = 1$. For large M, the distribution of Z can still be approximated by a Gaussian with

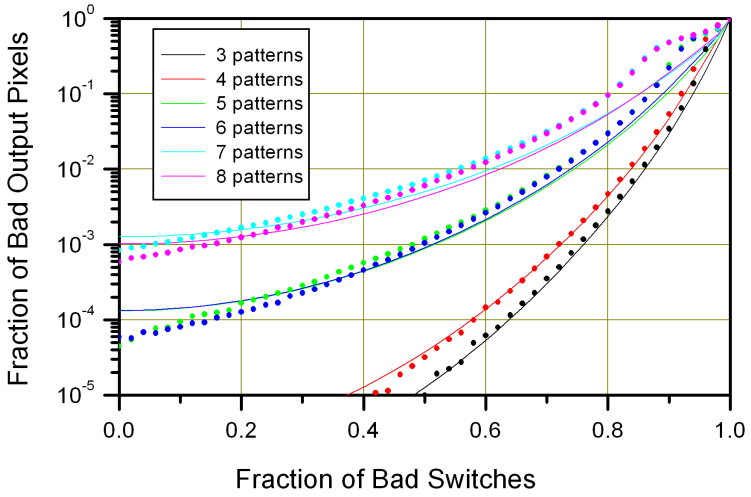


Figure 2.3: Defect tolerance of Crossnet as Hopfield network. The results are from a recurrent InBar as a Hopfield network: the fraction of wrong output bits as a function of the fraction of disconnected molecular devices (latching switches). Lines show the results of an approximate analytical theory (see [44]), while dots those of a numerical experiment.

mean $n(p_+ - p_-)$, and variance n.

Under the new criteria, the output is correct if $f(x_i) = \xi_i^0$. By comparing Eq. (2.5) with the stable solution Eq.(2.6) and talking into account the factor g, one finds that this means $Z > 1/g$. Therefore, we need to integrate the distribution of Z over $-\infty$ to $1/g$ to get the ratio of wrong bits. The results of theoretical calculation and experimental simulation are shown in Fig 2.3. The simulation was done for $N = 3744, M = 25, g = 1.0$. We can see that the Corruption Rate can reach as high as 80%, while still retain a stability of 99%.

2.4 Storing Correlated Patterns

The calculations in the previous section are based on the assumption that the patterns are independent to each other. The storage of correlated patterns in Hopfield model is a general topic addressed by many researches in the 1980s. We can define two types of correlation: the "semantic" correlation is that among different patterns, while the "spacial" correlation is among different cells of the network. The storage capacities for both types of correlation are calculated in [82] using the space of interactions method proposed by Gardner [83]. Phase diagram based on Ising Model for the spatially correlated patterns

are obtained in [84].

Semantic Correlation (simply refereed to as correlation in the following discussion) generally decrease the storage capability of the network. Basically, when stored patterns are correlated with each other, the noise distribution become biased and the simple Hebb rule is no longer suitable. Several other prescriptions with some modifications to Eq.(2.7) have been proposed. The most effective one is the Pseudo-Inverse prescription illustrated in [1], which gives capacity of $P_{\max} \sim N$. But nonlocality has to be introduced in the learning process. Alternative local learning rules are proposed in [85] and [86], which give a capacity of $\alpha_c = (1-|b|)^2/2\ln N$ for all the patterns to be stable, where $0 < |b| < 1$ is the level of correlation.

Here we try to use the same signal-noise analysis in the previous section to study how the correlation affects the storage capability of the network if the learning rule remains unchanged as Eq.(2.7).

We use the method proposed in [85] to generate the hierarchically correlated patterns. The probability distribution of the patterns sequences are:

1. Ancestors

$$p(\xi_i^\mu) = \frac{1}{2}(1+a)\delta(\xi_i^\mu - 1) + \frac{1}{2}(1-a)\delta(\xi_i^\mu + 1) \tag{2.19}$$

2. Descendants

$$p(\xi_i^{\mu\nu}) = \frac{1}{2}(1+\xi_i^\mu b)\delta(\xi_i^{\mu\nu} - 1) + \frac{1}{2}(1-\xi_i^\mu b)\delta(\xi_i^{\mu\nu} + 1) \tag{2.20}$$

where $\delta(x) = 0$ for $x \neq 0$ and $\delta(0) = 1$. The superscript $\mu\nu$ in Eq. (2.20) means that the descendent pattern ν is derived from ancestor pattern μ. We see that the ancestors are independent random sequences with a bias controlled by a. Later we will set $a = 0$ and only consider the unbiased patterns. The distribution of descendants are dependent on their ancestors, creating a correlation controlled by b among the patterns. The correlations among the descendants themselves, and that between the descendants and their ancestors are, respectively:

$$\begin{aligned}
\langle\langle \xi_i^{\mu\nu}\xi_i^{\mu'\nu'}\rangle\rangle &= b^2 \qquad (\mu = \mu', \nu \neq \nu') && (2.21)\\
&= a^2 b^2 \qquad (\mu \neq \mu') && (2.22)\\
\langle\langle \xi_i^{\mu\nu}\xi_i^{\mu'}\rangle\rangle &= b \qquad (\mu = \mu') && (2.23)\\
&= a^2 b \qquad (\mu \neq \mu') && (2.24)
\end{aligned}$$

where $\langle\langle\cdot\rangle\rangle$ means average over all pixels and all patterns.

The average bias of the patterns are:

$$\langle\langle \xi_i^{\mu} \rangle\rangle = a \tag{2.25}$$
$$\langle\langle \xi_i^{\mu\nu} \rangle\rangle = ab \tag{2.26}$$

Let us assume $a = 0$, i.e., there is no bias for any patterns.

We study two different cases. In the first one we try to store patterns that are descendants from a single ancestor. In the second case all patterns are from the same ancestor except the one we want to retrieve, which is from a different ancestor. Note that all the patterns stored in the network belong to the second generation. Intuitively, one expects worse results for the latter case because in that case the pattern we want to retrieve is alien to all the others. As usual, we define Z_i as:

$$Z_i = \sum_j \text{sign}(\xi_i^{\mu 0}\xi_j^{\mu 0} \sum_{\nu\neq 0} \xi_i^{\mu'\nu}\xi_j^{\mu'\nu}) = \sum_j \text{sign}(1 + \delta_{ij}), \tag{2.27}$$

where the noise term is

$$\delta_{ij} = \sum_{\nu\neq 0} \xi_i^{\mu 0}\xi_j^{\mu 0}\xi_i^{\mu'\nu}\xi_j^{\mu'\nu}. \tag{2.28}$$

Without losing generality, suppose the ancestor of the pattern we want to retrieve is $\mu = 0$, then the probability density function of the noise distribution will depend on four random variables: ξ_i^{00}, ξ_j^{00}, ξ_i^0, and ξ_j^0. However, since we are only interested in the summation over j in Eq. (2.27), we should calculate the density function averaged over ξ_j^{00}, and ξ_j^0. And because it does not matter if we calculate this average before or after the summations of probabilities in Eq. (2.35) and Eq. (2.36), we do it here to simplify the equations.

Therefore, for the the patterns with same ancestors ($\mu = \mu' = 0$), the probability distribution of the noise is:

$$\begin{aligned} f_\delta(x) = \quad & 2^{-P} C_{P-1}^{\frac{P-1+x\xi_i^{00}}{2}} (1 - b^4(\xi_i^0)^2)^{\frac{P-1-x\xi_i^{00}}{2}} \\ & \{(1 - b^2\xi_i^0)^{x\xi_i^{00}} + (1 + b^2\xi_i^0)^{x\xi_i^{00}} \\ & + b[(1 + b^2\xi_i^0)^{x\xi_i^{00}} - (1 - b^2\xi_i^0)^{x\xi_i^{00}}]\} \end{aligned} \tag{2.29}$$

And when $\mu \neq \mu'$ (let us assume $\mu' = 1$), the probability distribution of noise

is:

$$\begin{aligned} f_\delta(x) = \quad & 2^{-P} C_{\frac{P-1+x\xi_i^{00}}{2}}^{P-1} (1-b^4(\xi_i^1)^2)^{\frac{P-1-x\xi_i^{00}}{2}} \\ & \{(1-b^2\xi_i^1)^{x\xi_i^{00}} + (1+b^2\xi_i^1)^{x\xi_i^{00}} \\ & +a^2b[(1+b^2\xi_i^1)^{x\xi_i^{00}} - (1-b^2\xi_i^1)^{x\xi_i^{00}}]\} \end{aligned} \tag{2.30}$$

For large connectivity, Z_i obey Gaussian distribution with density function:

$$f_Z(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp[-\frac{[x - m(\xi_i^{00}, \xi_i^0, \xi_i^1)]^2}{2\sigma^2(\xi_i^{00}, \xi_i^0, \xi_i^1)}] \tag{2.31}$$

where the mean and variance are:

$$m(\xi_i^{00}, \xi_i^0, \xi_i^1) = 4M[p_+(\xi_i^{00}, \xi_i^0, \xi_i^1) - p_-(\xi_i^{00}, \xi_i^0, \xi_i^1)], \tag{2.32}$$

$$\sigma^2(\xi_i^{00}, \xi_i^0, \xi_i^1) = 4M[1 - p_0(\xi_i^{00}, \xi_i^0, \xi_i^1)], \tag{2.33}$$

and p_0, p_+, and p_- are defined as:

$$p_0 = f_\delta(-1), \tag{2.34}$$

$$p_+ = \sum_{x=0}^{P-1} f_\delta(x), \tag{2.35}$$

$$p_- = \sum_{x=-P+1}^{-2} f_\delta(x). \tag{2.36}$$

The mean and variance are all functions of random variables ξ_i^0,ξ_i^1,and ξ_i^{00}. They are, respectively, the ith pixels of the two ancestors, and the ith pixel of the first descendant (that is the one we want to retrieve). The distribution of Z_i need to be averaged over the distribution of those variables.

For the two cases, examples of the distribution of δ and Z are shown in Fig 2.4 and Fig 2.5, in which $a = 0, b = 0.3, 0.6, 0.9, M = 25, n = 10$. Simulation was done for $N = 1664, M = 25, g = 0.3$, the results was shown in Fig 2.6(a) and Fig 2.6(b).

As we can see, in both cases, the correlation between patterns has an adverse influence on retrieving. The correlation between the background patterns (those not being retrieved) will increase the noise amplitude, while the correlation between the target (the pattern that is being retrieved) and the other patterns will generate a biased noise distribution. Generally, error probability will increase in both situations. In the case of correlated patterns, however, the theory does not explain the experiment as well as that for independent

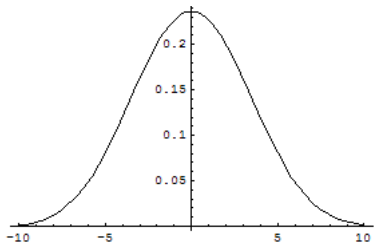
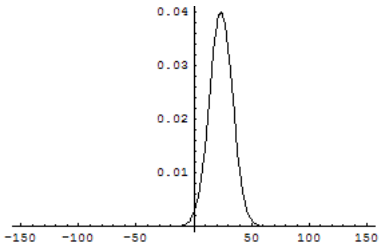
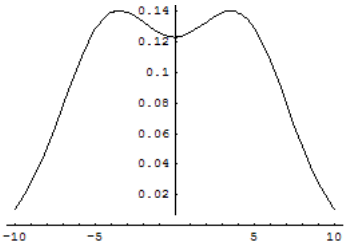
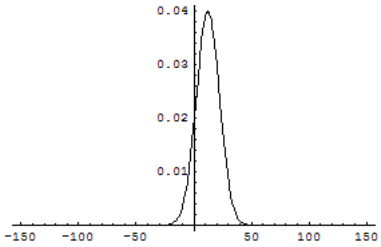
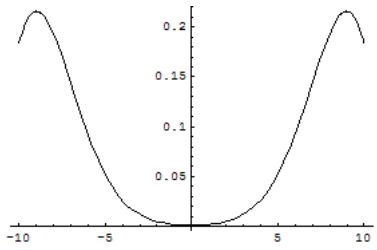
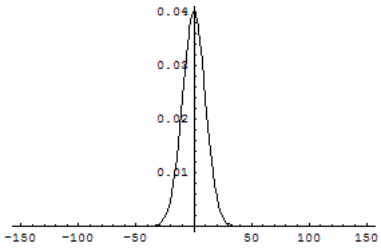
	Noise Distribution(δ)	Output Distribution(Z)
b=0.3		
b=0.6		
b=0.9		

Figure 2.4: Noise and output distribution example 1. All patterns are from the same ancestor except the first pattern.

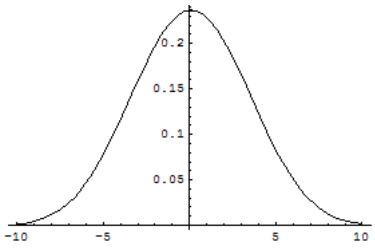
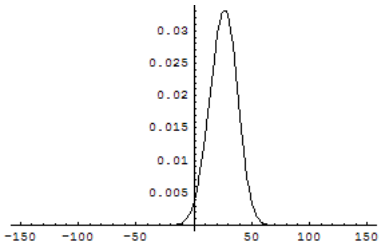
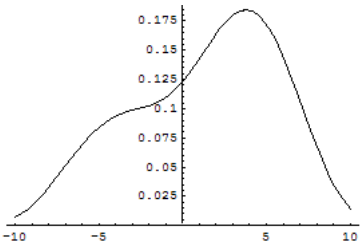
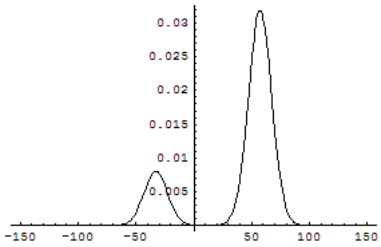
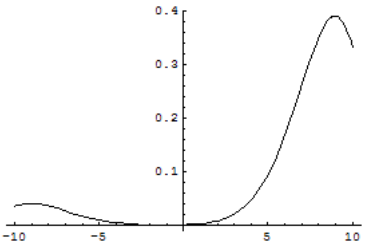
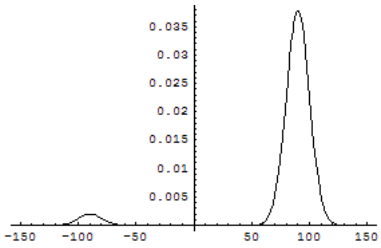
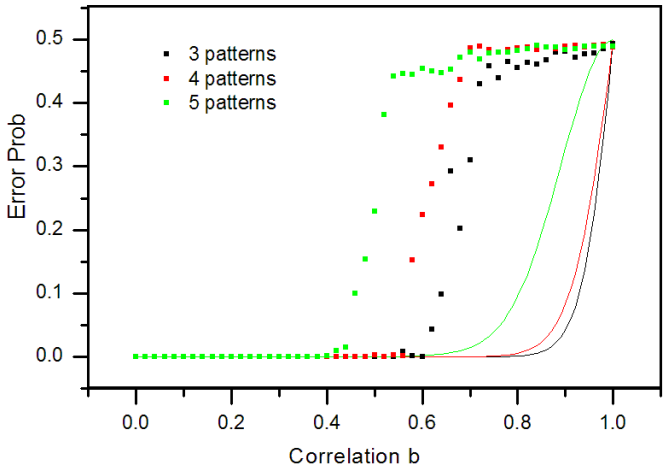
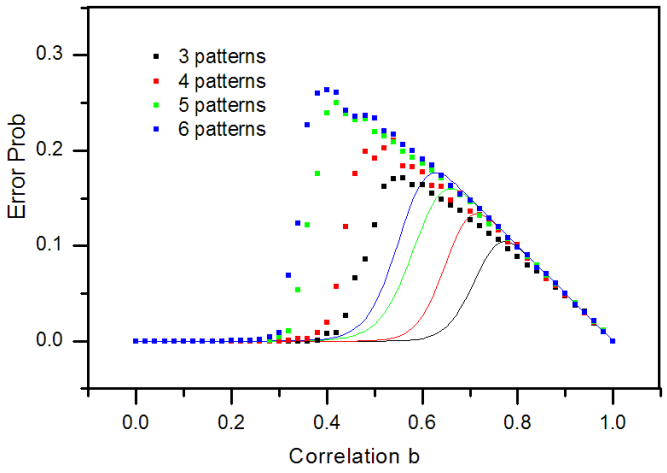
	Noise Distribution(δ)	Output Distribution(Z)
b=0.3		
b=0.6		
b=0.9		

Figure 2.5: Noise and output distribution example 2. All patterns are from the same ancestor.



(a)



(b)

Figure 2.6: Storing correlated patterns. a) All patterns are from the same ancestor except pattern 0. b) All patterns are from the same ancestor. The solid lines are theoretical results.

patterns. We believe the reason is that the Central Limit Theorem is no longer applicable if the patterns are not independent (see Eq. 2.31). Probably the distribution of Z_i is still close to Gaussian, but the variance of the sum of correlated random variables would certainly be larger than Eq. (2.33) has suggested. And that explains the worse performance than the theory has predicted.

Chapter 3

Introduction to Reinforcement Training

3.1 Supervised and Reinforcement Learning

In the field of machine learning, supervised learning means learning from examples. A training set with known correct outputs has to be provided for supervised learning. One way to teach a learning agent to achieve some desired input-output mapping (when the output of the learning agent is a function of the input and some internal parameters, for example), is to minimize the error by following the gradient of the error function with respect to the internal parameters. When the error function is a sum of squared differences between actual and desired outputs, for example, this technique is basically a Least Mean Square (LMS) fitting of the mapping function (which is usually a very complicated and nonlinear function) to the observed training data [87]. The most successful and widely applied training algorithm in neural networks, the error Back-Propagation (BP) method, is a gradient following technique [1]. It finds the gradient of the error function elegantly by back-propagating the error signals through the network itself. It can be extremely efficient because it allows the parallel computation power of the network to be fully utilized in the training stage. This method was invented independently several times [88–91]. Its implementation in CrossNets can be found in Ref. [61–63].

Supervised learning may be inadequate for the more general interactive learning because it is often impractical to obtain examples of desired behavior [92]. In reinforcement learning the learning agent learns through its own experience, by interacting with the "environment". It observe the current "state" of the environment s_t at time t; it then performs an "action" a_t to change the state to s_{t+1}, and at the same time a "reward" r_{t+1} is given as an evaluative

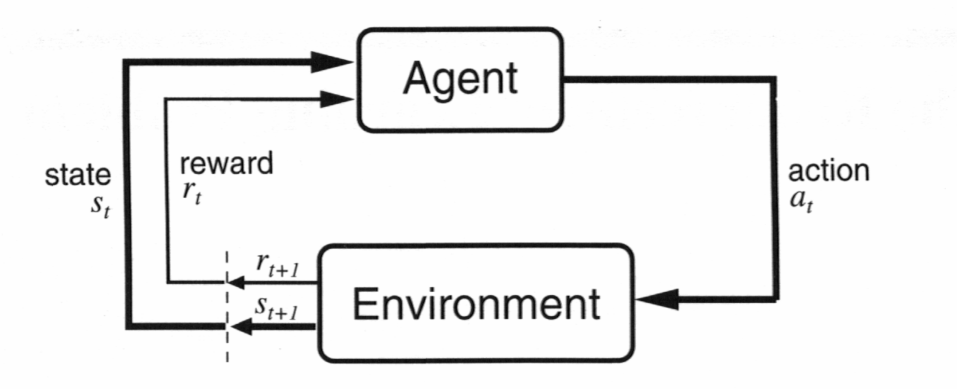


Figure 3.1: The agent-environment interaction in reinforcement learning (from Ref. [92]).

feedback from the environment (when the reward is negative it actually means punishment). This interaction is illustrated in Fig. 3.1. If such an interactive process does not depend on history, the reinforcement learning task is a Markov Decision Process [92].

The reward is intended to "reinforce" good behavior and punish bad behavior, like the way one would use snacks to train a dog. It is, however, very different from the supervision (such as the error function) provided to the agent in the supervised learning. First of all, the relation between current reward and the desired action is usually unknown to the agent (if there exists such a relation), while the error function is clearly defined as the difference between the actual and desired behavior. More importantly, it may be impossible to define the reward as a function of a single action. For example the reward may be "delayed", as in a chess game where the final consequence of wining and losing is a combined result of a sequence of actions taken by the player. In this case one would define the state to be the current positions of all the chessmen on the board. It is inappropriate to define the reward in a chess game for single actions, as these definitions are almost always shortsighted. It is also impractical to define reward as a function of a sequence of actions since the number possibilities is astronomical. To alow the reinforcement learning to solve this problem in full, the agent should be rewarded (punished) only at the final point when the game is won (lost) [92]. Note that in this case not only the learning agent are not given the correct answer, but the reward may not even be an indication of whether the action is right nor wrong - it is merely an indication of good or bad consequences. The agent may be receiving negative feedbacks while it is actually performing good actions, because some bad actions in an earlier time are causing the bad consequence. Under these

circumstances, the only way for the agent to learn the desired behavior is to try many different actions; and it is important that the goal is to maximize the long term expected reward.

3.2 General Reinforcement Learning Algorithms

The long term reward can be defined as the discounted future "return":

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{3.1}$$

where $0 \leq \gamma \leq 1$ is the discount factor. Discounting puts more weights on rewards from near future than from distant future; and it makes the summation finite as long as the reward is bounded. When $\gamma = 0$, it becomes a problem of "immediate reward".

The learning agent takes actions according to some "policy" which can be defined as the probability $\pi(s, a)$, of taking action a when current state is s. This is a general definition. If π can only take values 0 or 1, the policy is deterministic. In principle the agent can perform random search in the policy space and try to find an optimal policy that maximize the return. But the pure policy search is very inefficient when the reward is delayed.

Although in general the value for action can not (and should not) be defined, we can define "value" for a state (when a given policy is followed) to be the expected return starting from that state and following the policy thereafter:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\}, \tag{3.2}$$

where $E_{\pi}\{\cdot\}$ means the expectation under the condition that policy π is followed. The concept of value is very important in reinforcement learning. Crudely speaking, it guides the policy search toward more "promising" states (states with higher values) and actions (that leads to high-value states), and therefore achieve a much more efficient algorithm. Intuitively the value function is a much better guidance for improving policy than just the reward. In fact, when the values of stats are known for a given policy, a new policy that is greedy with respect to the value functions (i.e., a policy that deterministically leads to the state that has the highest value among possible choices), is guaranteed to be a better policy. This known as the "policy improvement terrorem" [93–95]. Based on this theorem, we arrive at the following learning algorithm:

1. evaluate the value functions for current policy;

2. improve the policy with an greedy policy based on the value functions;

3. repeat 1) and 2) until the policy is optimal.

This is know as policy iteration, and it is proven to converge to an optimal policy (a policy that maximize long term reward) [92]. It is the central idea lying behind all known reinforcement learning algorithms, for example, the Dynamic Programming (DP) [93], Mote Carlo (MC) methods [92], and Temporal Difference (TD) algorithms [96]. Note that the policy evaluation process itself is a multi-step iteration, but in practice it is not necessary to wait for its convergence. As a result the two processes can proceed in parallel. This is called value iteration [97]. But if that is the case, we may be using a crude estimations of the value functions in stead of their true values, therefore it is important to keep some degree of randomness in the policy (at least in the early stage of the learning) to prevent convergence into suboptimal solutions. There is an important topic of the trade off between the "exploration" (meaning random policy search) and the "exploitation" (meaning taking greedy policy derived from the estimated value functions) in reinforcement learning.

The TD algorithm, which performs the value evaluation, is of particular interest to us. It learns the value function of current state by comparing with that of the next state:

$$V(s) \leftarrow V(s) + r_{t+1} + \gamma V(s') - V(s), \tag{3.3}$$

where s is the current state and s' the next state. The learning rule uses the discounted value at next state, plus current reward, as the "target" to update the value of current state. The discount factor is the same as that in the definition of return (3.1). This learning rule is related to the famous Bellman's Equation in reinforcement learning [93] which expresses a relation between the value of a state and the values of its successor states. The idea behind this is simple: the states that lead to bad (good) states are also bad (good), unless the current reward r_{t+1} is exceptionally high (low).

To understand intuitively how the TD learning rule works, let us suppose we are trying to solve a delayed reward problem of finite episode, i.e., a problem in which a long sequence of actions are performed before a reward is received; and after the reward the sequence terminates and a new episode begins (an example can be found in Sec. 4.7). According to Eq. (3.3), the value functions of the terminal states (the states that directly leads to feedbacks) are the first to be learnt. This is because the values are initialized to zero, and Eq. (3.3) becomes $V(s_T) \leftarrow r_{T+1}$, where s_T denotes the last state, and r_{T+1} the reward at the end of the episode. After that, as the update rule is repeated for many episodes, the value function starts to "back-propagate" through the

chain of events, as stated by Eq. (3.3). If we are following a policy that leads to nonzero probability of visiting any state, the estimated value functions eventually converge to their true values [92].

The TD learning rule can be made more efficient by introducing the "eligibility traces", which remember which states that were visited during an episode. At each time step, all the states that have been visited are eligible for update by the following rule:

$$V(s) \leftarrow V(s) + \delta e_s(t), \tag{3.4}$$

where

$$\delta = r_{t+1} + \gamma V(s') - V(s), \tag{3.5}$$

as in Eq. (3.3), is called the TD error, and $e_s(t)$ is the eligibility trace of state s. Usually $e_s(t)$ is set to 1 each time the state s is visited and starts to decay exponentially thereafter with the factor $e^{-\lambda t}$. This algorithm is called TD(λ), where λ determines how much we emphasize on more recent states. It can be viewed as a combination of DP and MC methods [92]. It is often more efficient than simple TD (or TD(0)) method, because all states along the history of an episode are updated simultaneously, rather than just the current state. Justification of the method can be found in Ref. [92].

3.3 Generalization in Reinforcement Learning

The generalization ability is what differentiates supervised learning from simple memorization. After being taught the training examples, the agent has to make reasonable inductions about unseen samples. This is actually achieved by complicated interpolation and extrapolation from the examples by the input-output mapping function. Generalization in supervised learning for neural networks has been extensively studied [98]; and the results are very impressive [1].

The concept can be directly brought to reinforcement learning. In order to gain generalization ability in reinforcement learning, the value functions should have a parameterized function form $V(s, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a vector of parameters. The general gradient following techniques can be combined with TD method to learn such functions. If we know the "target" (correct answer) of $V(s, \boldsymbol{\theta})$, denoted by $v(s)$, the gradient following method tells us that the parameters should be updated according to the following:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta[v(s) - V(s, \boldsymbol{\theta})]\boldsymbol{\nabla}_{\boldsymbol{\theta}} V(s, \boldsymbol{\theta}), \tag{3.6}$$

where $\boldsymbol{\nabla}_{\boldsymbol{\theta}} \equiv \partial/\partial\boldsymbol{\theta}$, and $\eta > 0$ is the learning rate. Since we do not know the correct answer, we use the TD error instead. This (combined with the eligibility trace) yields the following algorithm:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\delta\mathbf{e}, \tag{3.7}$$

$$\mathbf{e} \leftarrow \gamma\lambda\mathbf{e} + \boldsymbol{\nabla}_{\boldsymbol{\theta}} V(s, \boldsymbol{\theta}), \tag{3.8}$$

where δ is the usual TD error (3.5), and the eligibility trace $\mathbf{e}$ here is a moving average of past gradients. For an analysis of convergence of this algorithm, please see Ref. [99].

Obviously, this method of function approximation can also be applied to the more general case where the state variable is vector of continuous values. A reinforcement framework that learns in continuous time and space can be found in Ref. [100]. In this case, the policy should also take a parameterized function form $\pi(\mathbf{s}, a, \boldsymbol{\theta})$. As we shall see later, this can be readily achieved by the neural network implementation of of policy improvement; and the statistical gradient following learning rules can be used to learn such parameterized policies.

The previous discussion helps us see the key difference between supervised and reinforcement learning from another angle. In supervised learning, the correct answers are already provided by the teacher, and the learning agent learns to replicate and generalize. In reinforcement learning, on the other hand, the agent has to find the answers first through the trial and error experience. In this sense the supervised learning only solves part of the problem while reinforcement learning with function approximation tries to solve the "whole learning problem" [92].

The most impressive applications of reinforcement learning are in the field of game playing. For example, Tesauro's TD-Gammon [101]. Other examples include control problems [102] and dynamic channel allocation [103].

3.4 Reinforcement Learning in Neural Networks and Actor-Critic Method

The focus of the following chapter is global reinforcement learning in ANNs. The term "global" emphasize the fact that the same reward r is used for update rules through the entire network. This is different from BP, in which the error function has to be back-propagated through the network to develop "local" error signals for weight update. Because no backward propagation of signals is required, reinforcement learning algorithms are believed to be more biologically plausible [104].

We discuss networks consisting of neural cells whose output signals y are sent to inputs of other cells through synapses with certain weights w_{ij}:

$$x_i = \sum_j w_{ij} y_j. \tag{3.9}$$

In this case, the policy can be achieved by the input-output mapping. More specifically, the state should be the input to the network; while the network output is the action. The synaptic weights are the internal parameters that determine the probability producing certain action under the condition of current state.

Most previous work on reinforcement training in neural networks has been focused on networks in which the randomness necessary for policy exploration is provided by stochastic neural cells ("Boltzmann machines" [105]) with random outputs. For such networks, Williams [106] has been able to derive a general class of "REINFORCE" (REward Increment = Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility) learning algorithms which achieve a statistical gradient ascent of the average reward, $E\{r|\mathbf{w}\}$, in the multi-dimensional space of w_{ij}. This gradient ascent is achieved with the help of signals readily developed in the network, similar to that in BP, but without back-propagating signals!

To apply reinforcement training to CrossNets, however, we need to extend the REINFORCE approach to networks with noisy synaptic weights. This is motivated by the characteristic of the synapsis in CMOL CrossNets. In CMOL hardware, the synapses are implemented by naoelectronics and therefore are by their nature stochastic (see Sec. 1.5). In addition, stochastic synapses are believed to be more biological [107, 108], and some existing supervised learning algorithms (see Refs. [109], [110] and [111] for examples) also rely on random weights. The generalization of REINFORCE to boarder sources of randomness and the derivation of some new learning rules are the topic of the following chapter.

The REINFORCE algorithms, however, including the learning rules we derived based this framework, are pure policy searching algorithms which does not use the value functions. These rules by themselves are only expected to work well for the "immediate reward" problems. The TD(λ) method we discussed in the previous section, on the other hand, is a learning rule for the value functions only. To solve the more general reinforcement tasks, we need to combine these two methods together to complete the full cycle of policy (or value) iteration. In Ref. [112] a set of learning rules called Value and Policy Search (VAPS) algorithms that combines policy and value search together were derived.

In this work, however, we adopt a method that is easier to implement for neural networks: the so-called actor-critic method. Two separate networks are needed in this method: the actor that performs the policy improvement (using reinforcement learning rules for nerual networks), and the critic that learns the value function through TD(λ) method combined with gradient following method. The gradient in this case can be readily obtained by usual BP method. For details please see Sec. 4.7.

Before deriving formal reinforcement learning algorithms for neural networks, however, in Sec. 4.1 we shows how the simple Hebbian adaptation (Sec. 1.5.2) can be combined with concept of global reinforcement to teach a recurrent network to solve parity problems. In Sec. 4.2 we will derive the REINFORCE approach from a more general point of view (the likelihood ratio method), so that it can be then applied to networks with other sources of randomness. (It will also be argued that the new derivation can be generalized to any feedforward or recurrent network.) In Section 4.3, 4.4 and 4.5 we derive novel learning rules for networks with random inputs or random weights, based on the arguments provided in Sec. 4.2. In Sec. 4.6 we apply those rules to a set of classification problems [1] and compare the results with those from other algorithms. In Sec. 4.7 we apply the new rules to a popular delayed reward control problem using the usual actor-critic method. Finally, in conclusion (Sec. 4.8) we discuss the advantages and limitations of the new learning rules.

[1] Classification problems can be formed as a special case of reinforcement learning problems.

Chapter 4

Reinforcement Algorithms for CrossNets

4.1 Reinforcement by Hebbian Adaptation

The initial idea of this approach has been based on the fact of chaotic excitation of recurrent CrossBars with differential dendritic signals (Eq. 1.12), at sufficiently large effective gain of somatic cells, as discussed in Sec. 1.4.1 (see Fig. 1.9). One may say that in this regime the system walks randomly though the multi-dimensional phase space of all possible values of somatic output voltages V_i.

We have also mentioned in Sec. 1.5.2 that the Hebbian (anti-Hebbian) adaptation (Eq. 1.29) strengthens (weakens) the input-output relation. In a recurrent network, Hebbian adaptation has an effect of stabilizing the system (and therefore suppressing the chaos); while the anti-Hebbian rule, has the opposite effect of driving the system away from current state (or even starting a chaotic excitation from a otherwise stable state). Fig. 4.1 shows an example of the effect of Hebbian (anti-Hebbian) adaptation for $\eta > 0$ ($\eta < 0$).

Now, let us see how chaotic excitation, utilized as exploration, can be combined with with the Hebbian adaptation as reinforcement to implement learning ability. First, input signals are inserted into some of the cells, and outputs picked up from a smaller subset of cells. The system is allowed to evolve freely, but this evolution is periodically interrupted, for brief time intervals Δt, by the application of somatic output voltages V_j of each cell back to its input dendritic wires. Simultaneously, the tutor applies to all synapses a global shift S corresponding to its satisfaction with the system output at this particular instant: $S < 0$ if the network output is correct and $S > 0$ if it is not. This operation results is a small change of average synaptic weights

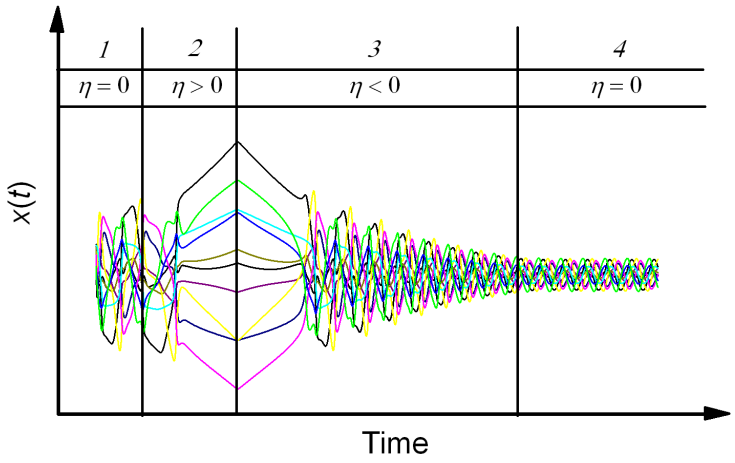


Figure 4.1: The effect of Hebbian and anti-Hebbian adaptation in a recurrent network. The figure shows the evolution of cell input signals x, as in Eq. (1.9). The system is a small (fully connected) recurrent network of 10 cells. It starts from a random carotic state as described in Sec. 1.4.1 and undergoes four stages of training. During each stage, the learning rate η is a constant and the simple Hebb's rule Eq. (1.29) is applied.

$\langle w_{ij} \rangle$, that is described by Eq. (1.28), thus implementing the Hebb rule if the system output is correct, and the anti-Hebb rule if it is incorrect. It had been our hope that the repeated application of this procedure would increase the probability of the system's eventual return to the "good" regions of the phase space, possibly with the eventual quenching of the chaotic dynamics.

Fig. 4.2 shows the results from a small recurrent InBar, taught by this technique to solve the parity problem. In a parity problem, the network is presented a binary vector (composed of $\pm$1s, for example); and it has to tell wether the number of +1s is even or odd. The parity function is actually a very difficult problem for machine learning, because it is extremely nonlinear. Indeed, by flipping just one bit in the input, one gets the opposite output. This is why the parity function (often its two-bit version, XOR) is popular for testing a learning algorithm's ability to do nonlinear classification.

The fact that this intuitive technique worked is actually quite surprising. It lacks theoretical support; and it is not expected to work for more complicated problems (like delayed reward problems). For those problems, we need the formal theoretical framework discussed in the following sections. However, since there are evidence suggesting that Hebbian kind of adaptation is really happening (and may be playing an important role) in the biological brain [113], this simple experiment may provide us a glimpse at how the brain functions.

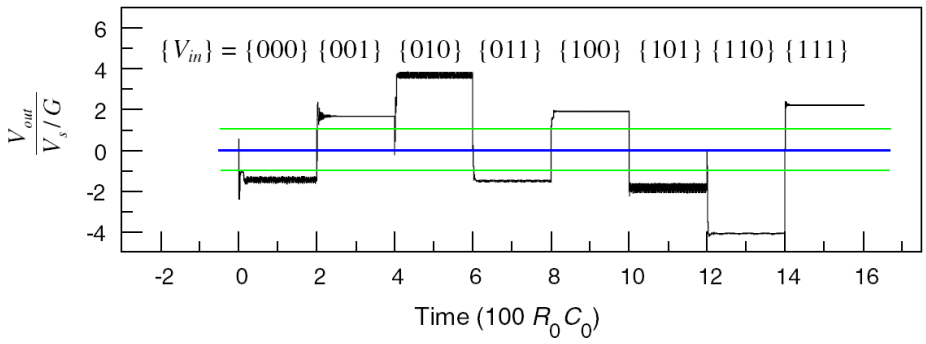


Figure 4.2: A preliminary result of reinforcement training: the input signal of the output cell of a recurrent InBar with quasi-continuous doubled synapses (Fig. 1.7(d)) trained to calculate parity of three binary inputs. (Any output signal value above the top horizontal line means binary unity, while that below the bottom line is binary zero). System parameters: $N = 612$, $M = 16$, $n = 10$, $R_L/(R/Mn^2) = 0.1$, $\Gamma_0/R_0C_0 = 5 \times 10^{-6}$, $g = 1$, $V_0/T = 10$, $S_{\max}/T = \ln 100 \approx 0.46$, $\Delta t = R_0C_0 = 10^{-3}$.

4.2 Derivation of the REINFORCE Algorithm using the Likelihood Ratio Method

Let $\mathbf{v} = \{v_1, v_2, ...\}$ denote a vector of some activity signals of a stochastic network with a set of deterministic internal parameters $\boldsymbol{\theta}$. Let us make a natural assumption that the probability $p(\mathbf{v}, \boldsymbol{\theta})$ of the system to generate a particular set of signals, at fixed network input, is a continuous function of $\boldsymbol{\theta}$. In this case the average reward received at a given set of parameters $\boldsymbol{\theta}$ is[1]

$$E\{r|\boldsymbol{\theta}\} = \sum_{\mathbf{v}\in U} r(\mathbf{v})p(\mathbf{v}, \boldsymbol{\theta}), \tag{4.1}$$

where we assume that the reward is some function $r(\mathbf{v})$ of the signals, and that U, which does not depend on $\boldsymbol{\theta}$, is the set of all possible vectors $\mathbf{v}$. By

[1]For the simplicity of notation, we assume that the signals take discrete values. All the results are trivially generalized to the case of continuous signals.

calculating the gradient with respect to $\boldsymbol{\theta}$ we obtain

$$\begin{aligned}\boldsymbol{\nabla}_{\boldsymbol{\theta}} E\{r|\boldsymbol{\theta}\} &= \sum_{\mathbf{v}\in U} r(\mathbf{v}) \boldsymbol{\nabla}_{\boldsymbol{\theta}} p(\mathbf{v},\boldsymbol{\theta}) \\ &= \sum_{\mathbf{v}\in U} r(\mathbf{v}) \frac{\boldsymbol{\nabla}_{\boldsymbol{\theta}} p(\mathbf{v},\boldsymbol{\theta})}{p(\mathbf{v},\boldsymbol{\theta})} p(\mathbf{v},\boldsymbol{\theta}) \\ &= E\{r\mathbf{e}|\boldsymbol{\theta}\}, \end{aligned} \tag{4.2}$$

where the vector

$$\begin{aligned}\mathbf{e} &= \frac{\boldsymbol{\nabla}_{\boldsymbol{\theta}} p(\mathbf{v},\boldsymbol{\theta})}{p(\mathbf{v},\boldsymbol{\theta})} \\ &= \boldsymbol{\nabla}_{\boldsymbol{\theta}} \ln[p(\mathbf{v},\boldsymbol{\theta})] \end{aligned} \tag{4.3}$$

had been known in classical statistics as the "score function" or "likelihood ratio", and was called "characteristic eligibility" in Ref. [106]. Equation (4.2) was originally proposed for computing performance gradients in i.d.d. (independent and identically distributed) processes [114].

If the incremental update rule is

$$\Delta\boldsymbol{\theta} = \eta r \mathbf{e}, \tag{4.4}$$

where η is positive constant ("learning rate"), then according to Eq. (4.2),

$$\begin{aligned} E\{\Delta\boldsymbol{\theta}|\boldsymbol{\theta}\} &= \eta E\{r\mathbf{e}|\boldsymbol{\theta}\} \\ &= \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} E\{r|\boldsymbol{\theta}\}. \end{aligned} \tag{4.5}$$

so that the average update is an unbiased estimate of the average reward gradient.

In the more general case of nonuniform learning rates,

$$\Delta\theta_k = \eta_k r e_k, \tag{4.6}$$

the estimation may deviate from the exact direction of the gradient. But as long as all components η_k of the vector η are non-negative and sufficiently small, we will always move "uphill" along the reward profile, because the

change of the average reward

$$\begin{aligned}\Delta E\{r|\boldsymbol{\theta}\} &\approx \boldsymbol{\nabla}_{\boldsymbol{\theta}} E\{r|\boldsymbol{\theta}\} \cdot E\{\Delta\boldsymbol{\theta}|\boldsymbol{\theta}\} \\ &= \sum_k \eta_k \left(\frac{\partial E\{r|\boldsymbol{\theta}\}}{\partial \theta_k}\right)^2 \\ &\geq 0. \end{aligned} \tag{4.7}$$

One can easily verify that this is also true for the case when coefficients η_k depend on time but do not correlate with re_k. (In that case we require $\langle \eta_k \rangle > 0$.)

This basic formulation is actually quite general. It does not tell us where the randomness comes from; and the complexity of the learning rule depends heavily on the form of the eligibility. The practical value of Eq. (4.6) is especially significant for the case of neural networks, where it is often possible to localize and simplify e_k. Before moving on, let us specify our notations for an MLP (multi-layer perceptron see Fig 1.3 [1]). We denote the collection of all weights in the network as $\mathbf{w}$, while the set of weights that connect mth layer to the previous layer is presented by vector $\mathbf{w}^m$. Similarly, the set of output signals of all the neurons in the network are denoted as $\mathbf{y}$, while $\mathbf{y}^m$ is the subset of outputs from layer m.

For an M-layer MLP with deterministic synapses and random cells, we can identify $\mathbf{v}$ with the set of all cell outputs $\mathbf{y}$, and $\boldsymbol{\theta}$ with weights $\mathbf{w}$. Note that for an MLP, the probability $p(\mathbf{y}, \mathbf{w})$ may be calculated layer by layer. Indeed, given that the output of layer $m-1$ is $\mathbf{y}^{m-1}$, the probability for the mth layer to produce output $\mathbf{y}^m$ is a function of $\mathbf{y}^m$, $\mathbf{w}^m$ and $\mathbf{y}^{m-1}$, i.e. $p^m = p^m(\mathbf{y}^m, \mathbf{w}^m, \mathbf{y}^{m-1})$. For the input layer, $p^1(\mathbf{y}^1)$ is simply the probability of a particular input $\mathbf{y}^1$ which does not depend on any weights or other cells. Therefore, according to the basic relation of conditional probability,

$$\begin{aligned} p(\mathbf{y}, \mathbf{w}) &= p^1(\mathbf{y}^1) p^2(\mathbf{y}^2, \mathbf{w}^2, \mathbf{y}^1) ... \\ &\quad p^M(\mathbf{y}^M, \mathbf{w}^M, \mathbf{y}^{M-1}). \end{aligned} \tag{4.8}$$

Now let us calculate the derivative of $\ln p(\mathbf{y}, \mathbf{w})$ with respect to a particular weight w_{ij} which connects cell j in layer $m-1$ to cell i in layer m:

$$e_{ij} = \frac{\partial \ln[p(\mathbf{y}, \mathbf{w})]}{\partial w_{ij}}. \tag{4.9}$$

By conditioning on (i.e. fixing) the previous layer, only one of the factors in

Eq. (4.8) is affected by the variation of w_{ij}. Therefore,

$$e_{ij} = \frac{\partial \ln[p^m(\mathbf{y}^m, \mathbf{w}^m, \mathbf{y}^{m-1})]}{\partial w_{ij}}. \tag{4.10}$$

Since $p^m(\mathbf{y}^m, \mathbf{w}^m, \mathbf{y}^{m-1})$ is simply a multiplication of independent probabilities for different cells in the mth layer (because there is no connections with a layer for a MLP), we can further "localize" e_{ij} to a single cell and obtain the following REINFORCE learning rule [106][2]:

$$\Delta w_{ij} = \eta_{ij} r e_{ij}, \tag{4.11a}$$

$$e_{ij} = \frac{\partial \ln[p_i(y_i, \mathbf{w}^m, \mathbf{y}^{m-1})]}{\partial w_{ij}}. \tag{4.11b}$$

As an example of how Eq. (4.11b) can lead to extremely simple learning rules, let us consider the "Bernoulli-logistic" stochastic cells. In this case, y_i can take only two values (0 and 1), with probabilities

$$p_i(y_i, \mathbf{w}^m, \mathbf{y}^{m-1}) = \begin{cases} 1 - g(x_i), & \text{if } y_i = 0, \\ g(x_i), & \text{if } y_i = 1, \end{cases} \tag{4.12}$$

where $g(x)$ is the "logistic" activation function,

$$g(x_i) = \frac{1}{1 + e^{-x_i}}. \tag{4.13}$$

It may be readily shown [106] that in this case e_{ij} takes the form

$$e_{ij} = (y_i - \langle y_i \rangle) y_j, \tag{4.14}$$

where $\langle y_i \rangle$ is the average output of cell i for a given input x_i. With all the learning rates equal, $\eta_{ij} = \eta$, Eq. (4.11a) yields a local rule called the Associative Reward-Inaction ($A_{r\text{-}i}$) [92]:

$$\Delta w_{ij} = \eta r (y_i - \langle y_i \rangle) y_j. \tag{4.15}$$

[2]Here we have used the same method of conditional probability to localize the learning rules. Williams's original derivation, however, started from $p(\mathbf{y}, \mathbf{w})$ with the assumption of random output and deterministic weights. By showing that Eq. (4.11a) can be viewed as a particular example of Eq. (4.4), and that the localization procedure only depends on the feed-forward structure of the network, we have removed the assumptions on the source of randomness.

The addition of a small extra term (which is not responsible for following the gradient but helps to kick the system out of local minima) yields the famous Associative Reward-Penalty ($A_{r\text{-}p}$) rule [115]:

$$\Delta w_{ij} = \eta \left[r(y_i - \langle y_i \rangle) y_j + \lambda(1-r)(-y_i - \langle y_i \rangle) y_j \right], \tag{4.16}$$

where $r \in [0,1]$[3] and λ is a small positive number.

Some other activation functions that lead to simple learning rules can be found in Ref. [116]. Existing algorithms that can be associated with the REINFORCE principle include also $L_{r\text{-}i}$ [117], and the learning rules for the "exponential families of distributions" [106]. All of them rely on stochastic neural cells.

For an arbitrary feedforward network the concept of "layer" is not very clear but still definable for the purpose of our derivation. The input layer is simply the collection of all those cells which receive only external inputs. The second layer then can be chosen from the rest of the cells (excluding those that are already categorized as the input layer) which receive no inputs except those from the input layer or external signal. Generally, we will assume that a cell belongs to layer m if all the cells that directly feed it belong to the previous layers and at least one of them belongs to layer $m-1$. This way, all the cells in the network can be labeled with a layer number, and $p(\mathbf{v}, \boldsymbol{\theta})$ can still be calculated "layer" by "layer". In this case, however, we should directly factorize $p(\mathbf{v}, \boldsymbol{\theta})$ into individual cells downstream through the connections rather than into layers because the input signal to any cell may come from any previous layer.

REINFORCE learning rule can even be generalized to recurrent networks. Williams derived "episodic" REINFORCE algorithms for episodic reinforce tasks, based on the fact that any recurrent network can be unfolded in time[1] into a feed-forward one (for details, see [106]). Baxter *et al.* [118] showed how to generalize REINFORCE algorithm for recurrent networks to non-episodic problems through Partially Observable Markov Decision Processes. All these extensions are applicable to the new derivation provided above.

[3]This is the reward representation originally proposed in Ref. [115]. According to the REINFORCE theory, however, the learning rule should be able to maximize the expectation value of r regardless of the representation. In our simulation $A_{r\text{-}p}$ performed equally well or even better for $r \in [-1,1]$ (Sec. 4.4).

4.3 Networks with Noisy Inputs

The merit of the derivation in the previous section is that it has much more flexibility to be easily grafted to networks with different sources of randomness. To show this, let us first consider a MLP in which an additive Gaussian noise with zero mean δx_i is injected into the cell input x_i:

$$x_i = \langle x_i \rangle + \delta x_i, \tag{4.17}$$

$$\langle x_i \rangle = \sum_j w_{ij} y_j. \tag{4.18}$$

In this case $y_i = g(x_i)$, where $g(x)$ can be an arbitrary activation function; while x_i is a random variable obeying the Gaussian distribution, with the following probability density function:

$$p_i(x_i, \mathbf{w}^m, \mathbf{y}^{m-1}) = A \exp\left[-\frac{(x_i - \langle x_i \rangle)^2}{\sigma_i^2}\right], \tag{4.19}$$

where A is a normalization factor and σ_i^2 is the variance of the random noise.

Now let us identify variables $\mathbf{v}$ of Sec. 4.2 with the set of input signals $\mathbf{x}$, and $\boldsymbol{\theta}$ with the set of synaptic weights w_{ij}. Then from Eq. (10), the eligibility component

$$e_{ij} = \frac{\partial \ln p_i}{\partial w_{ij}} = \frac{(x_i - \langle x_i \rangle) y_j}{\sigma_i^2}. \tag{4.20}$$

Therefore, the eligibility looks close to that expressed by Eq. (15), which leads to the $\mathrm{A_{r\text{-}i}}$ rule, even for an arbitrary activation function and for any (e.g., continuous) probability distribution of output signals y_j. As a result, if we take $\eta_{ij} = \eta \sigma_i^2$, we obtain the following simple local learning rule:

$$\textbf{Rule A1:} \quad \Delta w_{ij} = \eta r (x_i - \langle x_i \rangle) y_j. \tag{4.21}$$

which is very close in structure to $\mathrm{A_{r\text{-}i}}$.

Fig. 4.3(a) shows the learning dynamics (for 10 independent experiments) of a fully connected MLP (4-10-1) trained with Rule A1 to perform the parity function, which tells wether a binary vector contains even or odd number of 1s. The inputs are binary (-1 and +1).[4] The reward signal is simply $r = +1$ for the correct sign of the output signal, and $r = -1$ for the wrong sign. The network performance has been measured by the sliding average reward defined

[4]Because of such symmetric data representation, a certain number of bias cells with constant output (+1) had to be added to the input and hidden layer, in both this task, and that described in the next section. These biases are not included into the cell count.

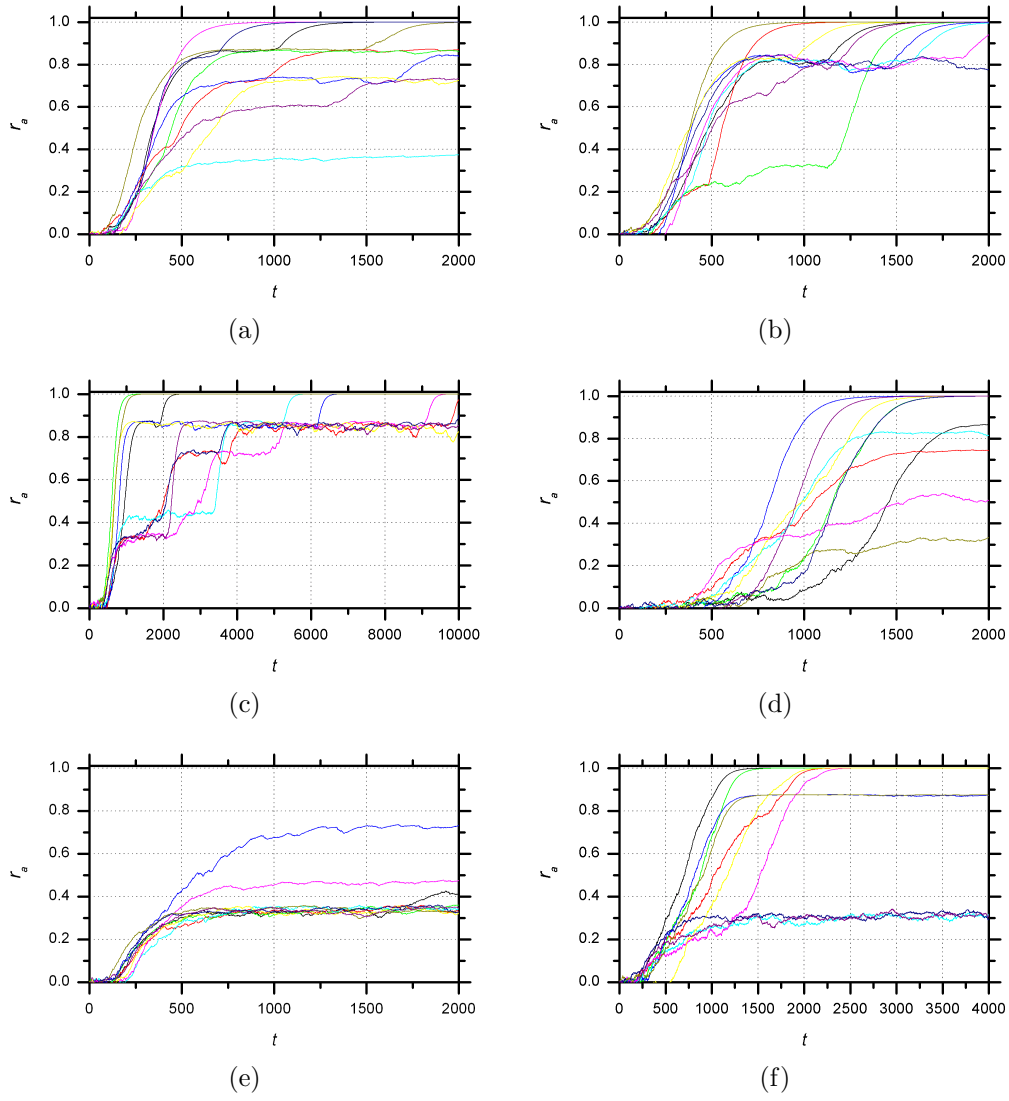


Figure 4.3: Parity function learning dynamics. a) A1, $\eta = 0.1$. b) A2, $\eta = 0.1, \lambda = 0.005$. c) A3, $\eta = 0.03, \lambda = 0.002$. d) B, $\eta = 0.02$. e) C1, $\eta = 0.4$, $\sigma(0) = 1$. f) C2, $\eta = 0.4$, $\sigma(0) = 1$, $\lambda = 0.005$.

as

$$r_a(t) = (1-\gamma)r_a(t-1) + \gamma r(t). \tag{4.22}$$

Here γ is a small positive constant (for the results shown in this paper, $\gamma = 0.01$), t is the training epoch number, and $r(t)$ is r averaged over all training patterns in the t-th epoch. One epoch consisted of the system exposure to all training patterns, and the resulting adaptation of all weights. The training set consisted of all 16 possible input patterns.

The neural cells were deterministic, with the following activation functions:

$$y_i = \tanh(h_i) = \tanh(\frac{G}{\sqrt{N_{m-1}}} x_i) \tag{4.23}$$

where $G = 0.4$ throughout the paper and N_{m-1} is the number of cells in the previous layer[5].

The noises δx_i are Gaussian random variables with equal fluctuation swings ($\sigma_i = \sigma$). In order to arrive finally at a trained network with fixed (deterministic) weights, we used the following "fluctuation quenching" procedure:

$$\sigma(t) = \sigma(0)[1 - r_a(t)]^\alpha, \tag{4.24}$$

where the constant α controls the quenching speed. This is in the line with the idea of balancing exploration and exploitation. In the simulation for parity function, $\alpha = 1$, and $\sigma(0) = 10$ unless otherwise specified.

Since Rule A1 looks very similar to the $A_{r\text{-}i}$ rule, (except that the post-activation signal y_i is replaced for the pre-activation signal x_i), it is natural to assume that the same problem of local minimum may be tackled by a similar antitrapping λ-term:

$$\textbf{Rule A2:} \quad \Delta w_{ij} = \eta \left[r(x_i - \langle x_i \rangle) + \lambda(1-r)(-x_i - \langle x_i \rangle) \right] y_j. \tag{4.25}$$

Learning dynamics for Rule A2 on the same parity problem is shown in Fig. 4.3(b). We can see clear improvement of performance in comparison with Rule A1.

[5]In simulations we have used the "normalized" input h_i in place of x_i for all the learning rules.

4.4 Networks with Stochastic Weights

To apply the derivation in Sec. 4.2 to the case when randomness comes from the weights, let us now consider an MLP composed of *deterministic* cells connected by *stochastic* synapses. In this case each w_{ij} in Eq. (A.1) is a random number. Let us assume that the synaptic weights have the Gaussian distribution with some mean value μ_{ij} and variance σ_{ij}^2. It can be easily shown that x_i is also a Gaussian random number with a pdf of Eq. (4.19).[6] But the mean and variance of x_i are now[7]:

$$\langle x_i \rangle = \sum_j \mu_{ij} y_j, \tag{4.26}$$

$$\sigma_i^2 = \sum_j \sigma_{ij}^2 y_j^2. \tag{4.27}$$

Therefore we obtain exactly the same learning rules (A1 and A2) if we calculate the derivative with respect to μ_{ij} and replace Δw_{ij} with $\Delta \mu_{ij}$. But one has to keep in mind that now the adaptive parameters are the average (or unperturbed) weights μ_{ij}.

There are two subtle difference between these two implementations of rules A. First, the variance σ_i^2 in the later is modulated by the activities in the previous layer - cf. Eq. (4.27). (This is exactly why it is, in this case, called "multiplicative" noise.) In most cases, however, the network quickly becomes saturated ($y_j \approx \pm 1$ for all j) and we have an almost constant variance. Second, by letting $\eta_{ij} = \eta \sigma_i^2$ the learning rate will depend on output signals if the noises come from the weights. However, since σ_i^2 is the "activity" of the previous layer weighted by σ_{ij}, for large networks it should largely remain constant or drift slowly with time. We believe (although we have not been unable to prove this analytically so far) that the correlation between $1/\sigma_i^2$ and re_{ij} is marginal. Therefore according to the arguments in Sec. 4.2, the learning rules should lead in the right direction. Of course we could have taken uniform learning rate $\eta_{ij} = \eta$ and completely eliminated both of those concerns with a more complex learning rule. But none of them have produced any noticeable differences in performance in our simulations.

In hardware implementation it may be difficult to keep track of the average input $\langle x_i \rangle$ (as in the case of CMOL [40]). In that case, the following rule can

[6] According to the central limit theorem, if the cell connectivity is sufficiently large, the distribution of x_i is approximately Gaussian regardless of the distribution of w_{ij}. In this case, our assumption of a Gaussian distribution of the weights may be dropped.

[7] Note that at this averaging, by our definition of p_i, y_j should be considered not as a random variable but a fixed number - cf. the derivation of Eq. (4.8).

be used as a substitute:

$$\textbf{Rule A3:} \quad \Delta\mu_{ij} = \eta\,[r(x_i - x_i') + \lambda(1-r)(-x_i - x_i')]\,y_j. \tag{4.28}$$

where x_i is as in Eq. (A.1),

$$x_i' = \sum_j w_{ij}' y_j, \tag{4.29}$$

w_{ij}, w_{ij}' are independent random weights with the same mean values, and r is the reward corresponding to w_{ij}. The proof is as the following.

Let us denote the weight changes derived from A2 and A3 respectively, as $\Delta\mu_{ij}$ and $\Delta\mu_{ij}'$. Apparently these weight changes are different random numbers, but let us calculate their expectation values (with $\lambda = 0$). First,

$$E\{\Delta\mu_{ij}'\} = E\{\eta r(x_i - x_i')y_j\}. \tag{4.30}$$

Since w_{ij} and w_{ij}' are independent, w_{ij}' does not correlate with r or y_j which are all functions of w_{ij}s. Thus we can replace all w_{ij}'s in Eq. (4.30) with their expectation values $\{w_{ij}'\} = \mu_{ij}$, this immediately gives us

$$\begin{aligned} E\{\Delta\mu_{ij}'\} &= E\{\eta r(x_i - \langle x_i\rangle)y_j\} \\ &= E\{\Delta\mu_{ij}\}. \end{aligned} \tag{4.31}$$

Therefore A1 and A3 follow the same gradient stochastically.[8]

Learning dynamics for Rule A3 on the same parity problem is shown in Fig. 4.3(c).

Rules A have been obtained by looking at the reward as a function of $\mathbf{x}$. However, there is another legitimate way to look at the reward: at a fixed network input, we may consider it a function of the synaptic weight set, $r = r(\mathbf{w})$. From this standpoint, in Eqs. (4.2)-(4) we can replace $\mathbf{v}$ with $\mathbf{w}$, and $\boldsymbol{\theta}$ with $\boldsymbol{\mu}$. Assuming the Gaussian distribution of the random weights,

$$p_{ij}(w_{ij}) = B\exp\left[-\frac{(w_{ij}-\mu_{ij})^2}{2\sigma_{ij}^2}\right]. \tag{4.32}$$

[8]Note that when implementing the second set of random weights (4.29), the output signals are memories from the first perturbation, i.e., $y_j = g(x_j)$. Otherwise (suppose if we had used $y_j' = g(x_j')$) we would not have been able to obtain the same expectation values.

we get

$$e_{ij} = \frac{\partial \ln p_i}{\partial \mu_{ij}} = \frac{w_{ij} - \mu_{ij}}{\sigma_{ij}^2}. \tag{4.33}$$

Again, we utilize the flexibility in choosing η_{ij} to further simplify the learning rule. With $\eta_{ij} = \eta\sigma_{ij}^2$, we obtain the following simple rule:

$$\textbf{Rule B:} \quad \Delta\mu_{ij} = \eta r(w_{ij} - \mu_{ij}). \tag{4.34}$$

This is perhaps the simplest learning rule suggested for artificial neural networks so far. Each weight change involves no information other than its own perturbation and the global reward signal. Fig. 4.3(d) shows that this rule follows the gradient at a lower speed than Rule A1 (4.21).[9] In the simulation the synaptic weights were independent Gaussian random variables with equal fluctuation swings ($\sigma_{ij} = \sigma$), but generally different mean values μ_{ij}. And the same quenching procedure as in the previous section has been applied.

An interesting fact is that rule B can also be applied to binary weights. For example if the weights w_{ij} can be either 0 or 1, with the probability

$$p(w_{ij}, \mu_{ij}) = \begin{cases} \mu_{ij}, & \text{if } w_{ij} = 1; \\ 1 - \mu_{ij}, & \text{if } w_{ij} = 0, \end{cases} \tag{4.35}$$

then

$$\frac{\partial \ln p}{\partial \mu_{ij}} = \begin{cases} 1/\mu_{ij}, & \text{if } w_{ij} = 1; \\ -1/(1 - \mu_{ij}), & \text{if } w_{ij} = 0. \end{cases} \tag{4.36}$$

Therefore

$$e_{ij} = \frac{w_{ij} - \mu_{ij}}{\mu_{ij}(1 - \mu_{ij})} = \frac{w_{ij} - \mu_{ij}}{\sigma_{ij}^2}, \tag{4.37}$$

and the learning rule is the same as Eq. (4.34).

4.5 Networks with Noisy Outputs

Let us consider an MLP in which an additive Gaussian noise with zero mean δy_i is directly injected into the cell output y_i:

$$y_i = \langle y_i \rangle + \delta y_i, \tag{4.38}$$

$$\langle y_i \rangle = g(x_i), \tag{4.39}$$

[9] Adding an antitrapping term, similar to those used in Rules $A_{r\text{-}p}$ and A2, does not help here.

where x_i is as in Eq. (A.1). In this case y_i is a random variable obeying the Gaussian distribution, with the following probability density function:

$$p_i(y_i, \mathbf{w}^m, \mathbf{y}^{m-1}) = C \exp\left[-\frac{(y_i - \langle y_i \rangle)^2}{\sigma_i^2}\right]. \tag{4.40}$$

And the eligibility component is

$$e_{ij} = \frac{\partial \ln p_i}{\partial w_{ij}} = \frac{(y_i - \langle y_i \rangle) g'(x_i) y_j}{\sigma_i^2}. \tag{4.41}$$

If we take $\eta_{ij} = \eta \sigma_i^2$, we obtain the following general learning rule:

$$\textbf{Rule C0:} \quad \Delta w_{ij} = \eta r (y_i - \langle y_i \rangle) y_j g'(x_i). \tag{4.42}$$

The simplest activation function one can think of is $g(x) = x$, only that it does not work for non-linear classification. It is tempting however to try the alternative segmentally linear activation function

$$\begin{cases} g(x) = x, & \text{if } |x| \leq 1; \\ g(x) = 1, & \text{if } |x| > 1, \end{cases} \tag{4.43}$$

using the following simple learning rule:

Rule C1:

$$\Delta w_{ij} = \begin{cases} \eta r (y_i - \langle y_i \rangle) y_j, & \text{if } |x| \leq 1; \\ 0, & \text{if } |x| > 1. \end{cases} \tag{4.44}$$

But the simulation results have shown that it does not work very well (Fig 4.3(e)).

Another option is to use $g(x) = \tanh(x)$, and $g'(x) = 1 - g^2(x)$. For this activation function we obtain (with the anti-trap term):

$$\begin{aligned} \textbf{Rule C2:} \quad \Delta w_{ij} = \eta [& r (y_i - \langle y_i \rangle) \\ & + \lambda (1 - r)(-y_i - \langle y_i \rangle)](1 - \langle y_i \rangle^2) y_j. \end{aligned} \tag{4.45}$$

The simulation results are shown in Fig 4.3(f).

Table 4.1: Generalization Performance for MONK's Problems

Algorithm		Problem 1	Problem 2	Problem 3	Parameters
Supervised Learning	BP	100	100	93.1	
	WDBP	100	100	97.2	
	Alopex	100	100	100	
	WP	100	100	93.5±1.3	$\eta = 0.1$
	SWNP	99.5±0.6	100	94.6±2.3	$\eta = 0.01$
Reinforcement Learning	$A_{r\text{-}i}$ (4.15)	77.2±0.2	76±12	96.8±0.0	$\eta = 0.1$
	$A_{r\text{-}p}$ (4.16)	99.4±0.5	99.8±0.3	96.8±0.0	differ for individual problems[a]
	Rule A1 (4.21)	79.2±2.5	77±11	96.7±1.3	$\eta = 0.1$, $\sigma(0) = 1$, $\alpha = 0$
	Rule A2 (A.5)	96.3±4.0	99.7±0.5	96.8±0.0	differ for individual problems[b]
	Rule A3 (A.5)	94.2±6.4	99.8±0.4	96.8±0.0	differ for individual problems[c]
	Rule B (4.34)	79.4±2.8	72.2±5.8	95.2±1.5	$\eta = 0.008$, $\sigma(0) = 1$, $\alpha = 0$
	Rule C2 (4.45)	80.7±0.2	76±14	94.7±0.8	$\eta = 0.3$, $\lambda = 0.003$, $\sigma(0) = 1$, $\alpha = 1$

[a]The best results were achieved at $\eta = 0.6, \lambda = 0.035$ for problem 1, $\eta = 0.7, \lambda = 0.025$ for problem 2, and $\eta = 0.07, \lambda = 0.001$ for problem 3.

[b]The best results were achieved at $\eta = 0.08, \lambda = 0.005, \sigma(0) = 1.8, \alpha = 0.4$ for problem 1, $\eta = 0.1, \lambda = 0.003, \sigma(0) = 1, \alpha = 0.2$ for problem 2 and $\eta = 0.007, \lambda = 0.004, \sigma(0) = 1, \alpha = 1$ for problem 3.

[c]The best results were achieved at $\eta = 0.06, \lambda = 0.005, \sigma(0) = 2.0, \alpha = 0.8$ for problem 1, $\eta = 0.04, \lambda = 0.003, \sigma(0) = 1.2, \alpha = 0.4$ for problem 2 and $\eta = 0.003, \lambda = 0.003, \sigma(0) = 1, \alpha = 1$ for problem 3.

4.6 Comparison on MONK's Problems

In this section we applied the new rules to some classification benchmark tasks. There tasks are best solved by supervised learning algorithms such as BP, but we use them as a initial test of generalization performance of the learning rules. Because reinforcement learning address the more general task of learning, a supervised learning task can also be formed as a reinforcement task. Like the parity function problems, we simply give positive feedback when the network produce correct classification and vice versa.

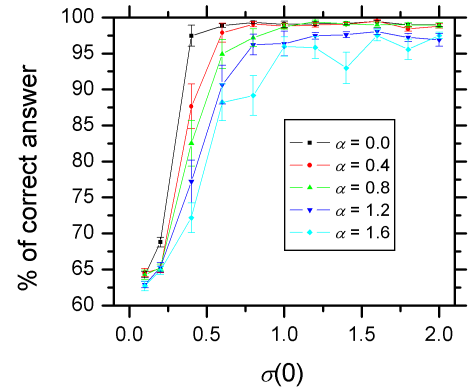
The "MONK's" problems [119] are widely used for neural network algorithm benchmarking. It contains three classification tasks with two classes. Each of the 3 problems contains 432 data vectors with 17 binary components each. For Problems 1, 2 and 3, there are, respectively, 124, 169, and 122 vectors in the training sets; the rest of the data are used as the test sets.

For the comparison of different training methods, we have used the MLPs of the same size as used earlier by other authors: 17-3-1. The positive output was treated as representing one class and negative one as representing the other class. The training was carried out in the online mode, i.e., the weights were updated after each pattern was presented at the input[10]. We have used $r = +1$ for correct classifications and $r = -1$ for wrong classifications[11]. Training was stopped either when the sliding average reward $r_a(t)$ exceeded 0.99 or after 10000 epochs.

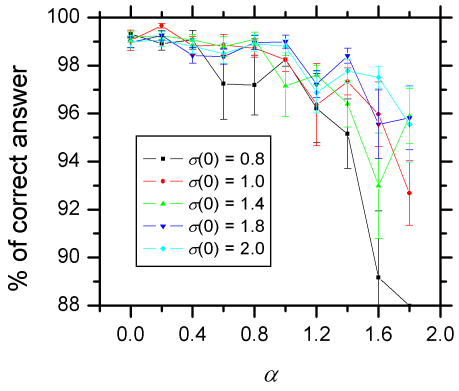
Table 4.1 shows the generalization performance (the percent of correct classifications on the test set) after training the networks with the new and some well known algorithms, including both supervised training rules (Error Backpropagation (BP) [119], Weight-Decay Backpropagation (WDBP) [121], Alopex [111], Weight Perturbation (WP) [110], Summed Weight Neuron Perturbation (SWNP) [109]) and global reinforcement rules ($A_{r\text{-}i}$ and $A_{r\text{-}p}$ [115]). Rule C1 was not tested on the MONK's problems due to its poor performance on the parity function problem. Rules A were implemented by additive noises in the inputs (multiplicative noises in the weights produce the same results but at slightly different parameters). The error bars correspond to the standard deviation of the results of 5 experiments except for the case of $A_{r\text{-}p}$, A2, and A3 where they were the results of 20 experiments. The simulation results for the first three algorithms have been borrowed from Ref. [111]. The last column of the table shows the parameters used for each training rule. In order to ensure a fair comparison between the best reinforcement learning rules ($A_{r\text{-}p}$, A2 and

[10] A rigorous analysis of the convergence of online stochastic algorithms can be found in Ref. [120].

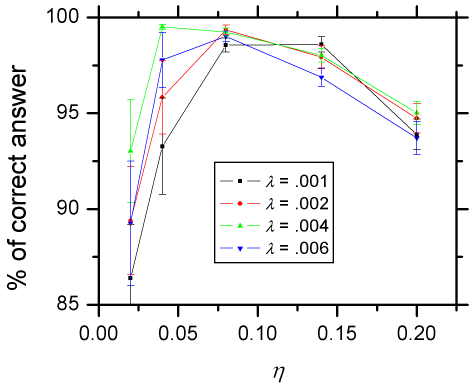
[11] $A_{r\text{-}p}$ was originally designed for $r \in [0, 1]$. But in our simulation we have found marginal improvement of performance under $r = \pm 1$



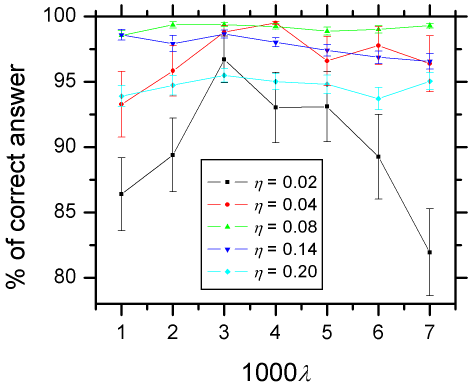
(a) A2 at $\eta = 0.1, \lambda = 0.003$.



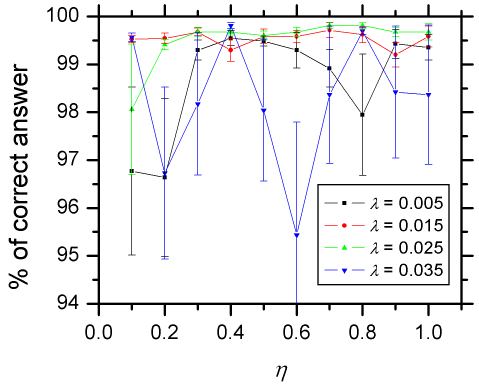
(b) A2 at $\eta = 0.1, \lambda = 0.003$.



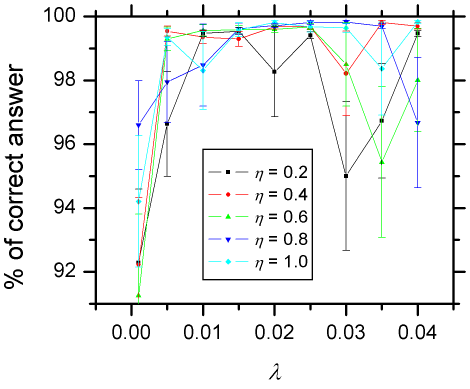
(c) A2 at $\sigma(0) = 1.0, \alpha = 0.2$.



(d) A2 at $\sigma(0) = 1.0, \alpha = 0.2$.



(e) $A_{r\text{-}p}$



(f) $A_{r\text{-}p}$

Figure 4.4: Optimization of the generalization performances with respect to learning parameters on the second of the MONK's problems. Results were averaged over 20 independent experiments and the error bar represents the standard deviation divided by $\sqrt{20}$. The values of $\sigma(0)$ and α in (c), (d) are the best values obtained from (a), (b).

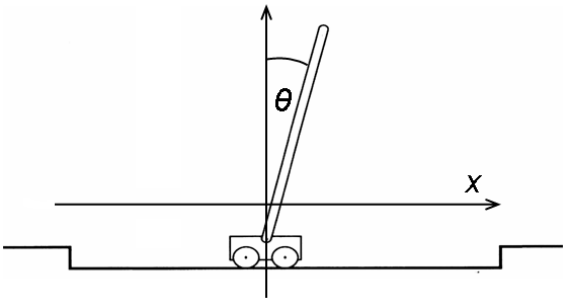


Figure 4.5: The Cart-Pole Balancing problem. The force applied to the cart is $a(t)F_{\max}$, where $-1 \leq a(t) \leq 1$ is a function of time. In our particular example $F_{\max} = 10$ N, the masses of the cart and pole are 1.0 kg and 0.1 kg, respectively, and the length of the pole is 1 m. The dynamics of the system is simulated with a time step of 0.02 s which is small in comparison with the dynamics time scales (which are of the order of 1 s).

A3), we have optimized their performances for individual problems over different learning parameters. Fig. 4.4 shows as an example the optimization on the second problem.

4.7 Comparison on the Cart-Pole Balancing Problem

One may argue that the global reinforcement rules have to be also characterized on problems which do not allow direct supervision. We have done this for the Cart-Pole Balancing task [102] in which the system tries to balance a pole hinged to a cart moving freely on a track (Fig. 4.5) by applying a horizontal force to the cart. A failure occurs when either the pole incline angle magnitude exceeds 12 degrees, or the cart hits one of the walls ($x = \pm 2.4$ m). A reward of $r = -1$ is issued upon failure and $r = 0.1$ otherwise[12]. This is a classic example of delayed reward problem. The agent is punished only when the pole actually falls, even though the actions responsible for the failure happened some time earlier.

To solve this problem, the usual actor-critic method [92] was used. The actor is a 4-30-1 MLP which takes the state vector of the cart-pole system $\{x(t), \dot{x}(t), \theta(t), \dot{\theta}(t)\}$ as input and produces a single output $a(t)$ as the action. This network has been trained by either A1 or $A_{r\text{-}i}$ (for this task, anti-trapping

[12]We have used $r = 0.1$ rather than $r = 0$ for intermediate movements to help jump start the learning process.

terms are not necessary) with the temporal difference (TD) error [92]

$$\delta(t) = r(t+1) + \gamma V(t+1) - V(t), \tag{4.46}$$

playing the role of the instant reward signal. In Eq. (4.46), $r(t+1)$ is the real reward from time t to $t+1$, $V(t)$ is the value function and γ is the discount factor. The TD error (Eq. 4.46) can be viewed as an evaluation of the long-term reward which truly reflect the goodness of current action. Therefore, by replacing r with $\delta(t)$ in the learning rules, the delayed reward problems can be solved. For example, in the case of TD(λ), the A1 rule takes the form

$$\Delta w_{ij}(t) = \eta_a \delta(t) e_{ij}(t), \tag{4.47a}$$

$$e_{ij}(t) = \gamma\lambda_{\mathrm{TD}} e_{ij}(t-1) + [x_i(t) - \langle x_i(t)\rangle] y_j(t). \tag{4.47b}$$

For Rule B, on the other hand, we would use the following eligibility:

$$e_{ij}(t) = \gamma\lambda_{\mathrm{TD}} e_{ij}(t-1) + [w_{ij}(t) - \mu_{ij}(t)]. \tag{4.48}$$

One more option here is to use an additional adaptation of fluctuation intensity instead of global quenching used in the previous tasks. Indeed, by identifying the set of variances σ_i with $\boldsymbol{\theta}$ in Eqs. (4.11) and letting $\eta_i = \eta_\sigma \sigma_i^2$, one arrives at the following[13]

$$\textbf{Rule } \sigma\textbf{:} \quad \Delta\sigma_i = \eta_\sigma r[(x_i - \langle x_i\rangle)^2 - \sigma_i^2]/\sigma_i. \tag{4.49}$$

Unfortunately, this rule seems inconvenient for hardware implementation.

The critic is a 5-30-1 MLP which takes the state-action vector $\{x(t), \dot{x}(t), \theta(t), \dot{\theta}(t), a(t)\}$ as input and produces a single output $V(t)$ as a value function estimate. The critic has been trained by error backpropagation with TD error. In the case of TD(λ),

$$\Delta\mathbf{w}(t) = \eta_c \delta(t) \mathbf{e}(t), \tag{4.50a}$$

$$\mathbf{e}(t) = \gamma\lambda_{\mathrm{TD}} \mathbf{e}(t-1) + \boldsymbol{\nabla}_{\mathbf{w}} V(t), \tag{4.50b}$$

where the gradient can be obtained by back-propagating the TD error through the network (for details, please see Ref. [1, 92]).

All the somatic cells in the critic network have the tanh activation function (4.23), except for the output cell which is linear:

$$V = y_i = 0.1 h_i. \tag{4.51}$$

[13] Actually, this rule had been derived by Williams [106] for random somas with the Gaussian statistics of the output signal fluctuations.

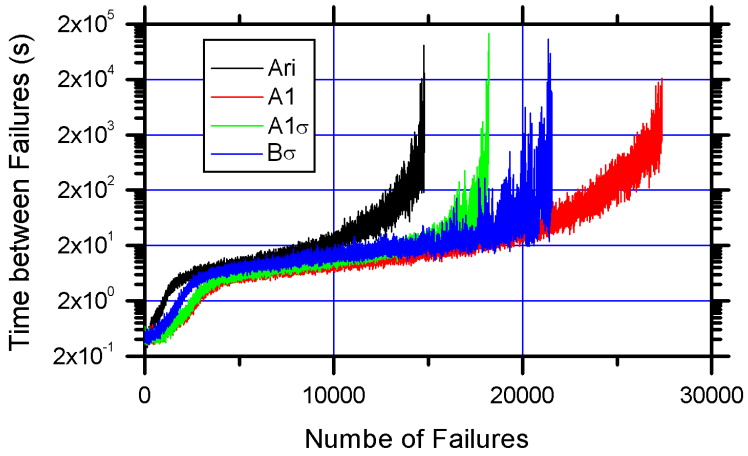


Figure 4.6: Training dynamics for the cart-pole balancing, using Rules A1, A1σ, B, and $A_{r\text{-}i}$. All results were averaged over 20 independent experiments. After each failure, the system is restored to its initial condition ($x = \dot{x} = \theta = \dot{\theta} = 0$), and the experiment is continued. Parameters used in the training are: for Rule A1: $\eta_a = 0.01$, and $\sigma_i = 10$ for all cells (no quenching); for Rule B: $\eta_a = 0.006$; for Rule σ: $\sigma_i = 10$ initially and $\eta_\sigma = 0.0002$; for Rule $A_{r\text{-}i}$: $\eta_a = 0.02$; for Backprop: $\eta_c = 10$; for TD(λ): $\gamma = 0.95$, $\lambda_{\rm TD} = 0.6$.

The results of simulation are shown in Fig. 4.6. As we can see, although rules $A_{r\text{-}i}$ and A1σ (which is a combo of A1 and σ rules) lead to faster training, simple rule A1 and B[14] are also able to fully solve this problem (i.e., to learn how to balance the pole without failure indefinitely) eventually. Rule B, in particular, is extremely simple. This rule does not assume any knowledge of the structure of the network, therefore it is applicable to any learning model with an arbitrary set of internal parameters (not limited to neural networks).

In comparison with the usual reinforcement learning using RBF network [92] or CMAC [122], the learning is slow. However, unlike in those methods, rule A1 learns directly in the continuous space. (No discretization whatsoever is involved). We believe it makes our method applicable to a broader range of tasks.

4.8 Discussion

Although the performance of the global reinforcement training rules on the classification tasks are not better than that of supervised learning rules, their

[14]Combined with rule σ Rule B performed slightly better.

natural domain of application are cases when the supervision is not available. An interesting fact, however, is that reinforcement learning rules (including our new rules) all had very good performance on the third of the MONK's problems, which includes noise in the training set and had been believed to be the most difficult one for supervised learning[119].

The most important result of this study is a constructive proof that neural networks with stochastic synapses, can perform classification (trained using at least one rule family A), and reinforcement tasks (trained using rule A and B) on a par with the Boltzmann machines using $A_{r\text{-}i}$ and $A_{r\text{-}p}$ rules. The new rules are very simple and local, giving hope that it may be readily implemented in hardware, in particular in ultradense nanoelectronic networks like CMOL CrossNets [40]. Such implementation is the topic of the next chapter.

Note that although rules of types A and B can be derived for the same random system using the same approach, they are rather different in structure. Indeed, the weight changes $\Delta\mu_{ij}$ given by these rules are different random numbers, even though for $\eta_{ij} = \eta$ they have the same expectation value $\langle\Delta\mu_{ij}\rangle = \eta\partial E\{r|\boldsymbol{\mu}\}/\partial\mu_{ij}$. Apparently Rule B is ultimately localized and makes less use of information about the network. (Indeed, the structure of Eq. (4.34) completely ignores the existence of the pre-synaptic and post-synaptic signals!). We believe that this is why Rules A1 and A2 are more effective for training MLPs, while Rule B may be more applicable to learning in more complex systems. (Our plans are to explore this opportunity.) The two implementations (either by additive or multiplicative noises) of rules A, on the other hand, provide flexibility for hardware implementation.

Rules C do not perform as well as rules A even with the anti-trap term. We believe that the advantage of adding noise in the input is because that the squashing activation function automatically quenches the noise when the cell is saturated. We have tried, instead of using uniform noises, to manually quench the noise in rules C according to the cell saturation, i.e., $\sigma_i(t) = \sigma(t)g'(x_i)$, and have obtained results comparable to those of rules A. However, we did not include the discussion about this "modulated noise in the outputs" in this paper, because we believe that rather than a new set of learning rules, it is more of an awkward way of implementing rules A.

Chapter 5

In Situ Reinforcement Training

In this chapter we discuss hardware implementation of the learning rules discussed in the previous chapter. We use the stochastic multiplication method, using time-division multiplexing (TDM) [62], as discussed in Sec. 1.5.3, for the in-situ implementation of the synaptic weight change proportional to the product of two voltages $\Delta\langle w_{ij}\rangle = \eta V_i V_j$. This method may be applied, for example, to pre-synaptic and post-synaptic signals (giving the Hebb's rule), or for the realization of error backpropagation (in that case one of the voltages represents the feedforward signal while another one, the backpropagating error). Superficially, it might seem that this approach enables one to implement any of rules A1-A3 readily. Indeed, Δw_{ij} given by any of these rules may be presented as a sum of either two (A1) or four (A2 and A3) signal products. Thus, using either two or four composite synapses in parallel (which is quite natural in the CrossNet topology [40, 45]) may apparently do the job.

Two complications, however, cause our usual in-situ training method fail to work for reinforcement learning. The first one is that Eq. (1.37) has a relaxation term. It makes the in-situ learning rule close to Oja's rule $\Delta\langle w_{ij}\rangle = V_i V_j - b\langle w_{ij}\rangle$, with $b > 0$ [123]. This relaxation has an effect of constraining the growth of synaptic weights. It is useful in unsupervised learning [1] like Principle Component Analysis (PCA) [1], but it is detrimental to BP training when the training set is very large [63]. For reinforcement learning rules, the result is even worse [65]. In CrossNets this term comes from the fact that the switches that are already in the ON state can no longer be turned on, while random fluctuations in the opposite direction can easily change the state to OFF. For deterministic updates, this naturally limits the range of weights within the capability of the hardware. For noisy updates such as those in reinforcement

[1] The unsupervised learning has even less external guidance than reinforcement learning: neither supervision or any kind of feed back is available. The objective in unsupervised learning, for example, can be discovering some underling structure of the data [124].

learning rules, however, this results in a limitation that is less than the full capacity of the synapses composed of arrays of switches (see the next section for details).

The other (and more important) problem is the internal stochasticity of the learning rules as such - see previous chapter - resulting from the very nature of reinforcement learning which implies a statistic exploration of the phase space. As a result, many weight iterations are needed to average out the noise. In the case of continuous weights, this means just more training time. For the discrete weight learning rules, however, large noise can result in a complete failure of training, even after the relaxation effects have been. Indeed, discrete weights jump in relatively big steps; therefore the average (correct) direction of their evolution is difficult to calculate. Smaller learning rate could help to average the noise and give a smoother learning curve; however, sufficiently large rate is necessary for any gradient following learning rule to avoid local optima.

The following sections try to quantify these effects and propose a solution that solves both problems.

5.1 Relaxation Term in In Situ Training

Let p be the probability of a group of CMOL switches to be in the "on" state. The dynamics of the probability adaptation is (1.21)

$$dp/dt = \Gamma_{\uparrow}(1-p) - \Gamma_{\downarrow}p. \tag{5.1}$$

In the following let us assume that the tunnelling rates are determined as follows

$$\begin{cases} \Gamma_{\uparrow} = \Gamma_0 dw, \Gamma_{\downarrow} = 0, & \text{if } dw \geq 0; \\ \Gamma_{\uparrow} = 0, \Gamma_{\downarrow} = -\Gamma_0 dw, & \text{if } dw < 0. \end{cases} \tag{5.2}$$

where dw is the incremental weight change derived from the learning algorithm and Γ_0 is a constant.

For consistent updates (i.e., updates in which dw never change sign), Eq. (5.2) ensures that only one of the tunnelling rates $\Gamma_{\uparrow}$ and $\Gamma_{\downarrow}$ is nonzero. When the learning rule is noisy, however, this update rule has an intrinsic problem that is independent of the learning algorithm or the discreteness of the weights. Fig. 5.1(a) shows an example of this problem. The goal of the experiment is to try to drive the weight to saturation ($p = 0, 1$) using noisy weight change with constant mean. The incremental weight change dw in this case is a Gaussian random number with mean value m and variance v^2. Even though the constant mean is always driving the weight toward $p = 0$ or 1, depending on the sign of

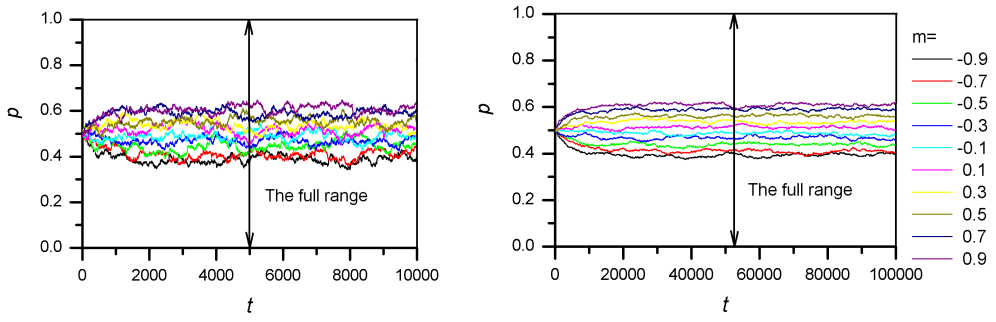


Figure 5.1: Noisy update and relaxation. a) Dynamics of p with different m. Other parameters: $v = 5$ and $\Gamma_0 \Delta t = 0.0006$, where Δt is the time interval used in the simulation. b) Dynamics of p with 10 times smaller learning rate, $\Gamma_0 \Delta t = 0.00006$, but 10 times longer training time. Other parameters are the same as a).

m, the random fluctuation has the opposite effect of driving the wight toward neutral (i.e., $p = 0.5$). At the equilibrium, only a fraction of the switches can be turned on no matter how long we apply the adaptation.

This problem can not be solved by reducing the the tunnelling rate and therefore "smoothing out" the fluctuation. For example, in Fig 5.1(b) the curves are smoother, but they have the same saturation point as those in Fig 5.1(a). A quantitative explanation is provided by the following.

For simplicity let us consider the case of very small learning rates. If the time scale of the dynamics of p is much larger than the time scale of the random fluctuation, dynamic rule Eq. (5.1) can be written as

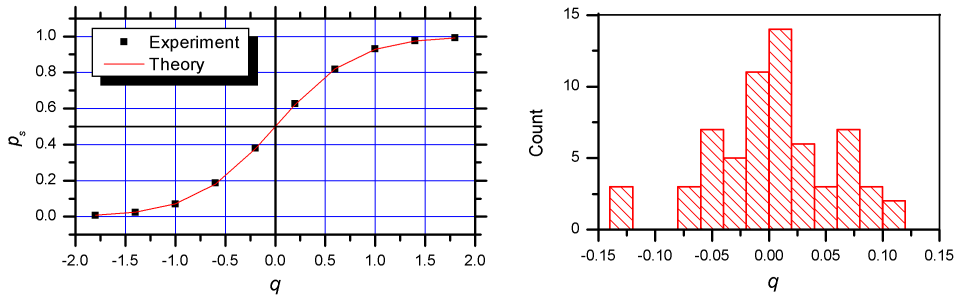
$$\dot{p} = \langle \Gamma_\uparrow \rangle (1 - p) - \langle \Gamma_\downarrow \rangle p,$$

where $\langle x \rangle$ denotes the mean value of the random variable x. From this we immediately obtain the saturation point at $\dot{p} = 0$:

$$p_s = \frac{\langle \Gamma_\uparrow \rangle}{\langle \Gamma_\uparrow \rangle + \langle \Gamma_\downarrow \rangle}. \tag{5.3}$$

From Eq. (5.3) we can see that if one and only one of the two values $\langle \Gamma_\uparrow \rangle$ and $\langle \Gamma_\downarrow \rangle$ is nonzero, the weight will be able to reach maximum potential, i.e., saturating at either 1 or 0. Otherwise, the full potential of the weight can never be reached.

For the Gaussian weight change in the previous section, it can be calculated



(a) (b)

Figure 5.2: Quantitative explanation of relaxation. a) Saturation point under Gaussian noisy update. In this experiment $\Gamma_0 \Delta t = 0.00003$, $v = 5$ and the values of p_s were taken at $t = 200000$. b) Histogram of q for A2 training for the firs of the MONK's problems. The plot shows the result for all the weights under one exposure to all the patterns in the training set. Parameters used in A2: $\eta = 0.0008, \lambda = 0.005, \sigma(0) = 1.8, \alpha = 1$.

that

$$p_s(q) = \frac{\sqrt{2}e^{-q^2/2} + \sqrt{\pi}\, q + \sqrt{\pi}\, q \,\mathrm{erf}\left(q/\sqrt{2}\right)}{2\sqrt{2}e^{-q^2/2} + 2\sqrt{\pi}\, q \,\mathrm{erf}\left(q/\sqrt{2}\right)},$$

where $q = m/v$. In Fig. we plot p_s against q and compare with experiment results.

The implication of the this formula for training is the following. In order for the full potential of the weights to be reached, the "signal to noise ratio" $|q|$ has to be at least larger than 1. Fig. 5.2(b) shows the distribution of q for the weights in A2 training for the MONK's problem. As we can see, in this case the learning rule is too noisy for the hardware update rule Eq. (5.1) to work. In fact, for $q \sim 0.1$, only about 10% of the weight range is being used. In other words, we need ten times more hardware resources to achieve the same capability.

5.2 Solution

Fortunately, this problem can be solved in the following way. Let us use a separate (relatively small) array of switches, whose total synaptic weight represents Δw_{ij}, to perform the stochastic multiplication of pre- and post-synaptic signals. The results are picked up, added to calculate the aggregate w_{ij}, and then written into another (typically, larger) array of n binary switches,

which represents the full synaptic weight. (A somewhat similar trick was used for supervised training in Ref. [61], with good results). Since in this implementation all the switches in the multiplication array are reset to ($p \approx 0.5$) at each iteration, the relaxation term disappears. On the other hand, weights (represented by the secondary arrays) are able to reach their maximum values, because the weight changes are simply added to the original weights.

Suppose the synapses are composed of two arrays of switches (one for positive weights, and the other for negative weights): $w_{ij} = w_{ij}^{+} - w_{ij}^{-}$, where $0 \leq w_{ij}^{\pm} \leq n$ are integers, and that each group has a corresponding multiplication unit with just one binary switch. Let $dw = \Delta w_{ij} < 1$ be the weight change calculated from the learning rule; then using the method discussed above we can implement the following discrete learning rule in CrossNet hardware:

$$w_{ij}^{+} \longleftarrow \begin{cases} w_{ij}^{+} + 1, & \text{with probability } dw \text{ if } dw > 0 \text{ and } w_{ij}^{+} < n; \\ w_{ij}^{+} - 1, & \text{with probability } -dw \text{ if } dw < 0 \text{ and } w_{ij}^{+} > 0; \\ w_{ij}^{+}, & \text{otherwise.} \end{cases} , \quad (5.4a)$$

$$w_{ij}^{-} \longleftarrow \begin{cases} w_{ij}^{-} - 1, & \text{with probability } dw \text{ if } dw > 0 \text{ and } w_{ij}^{-} > 0; \\ w_{ij}^{-} + 1, & \text{with probability } -dw \text{ if } dw < 0 \text{ and } w_{ij}^{-} < n; \\ w_{ij}^{-}, & \text{otherwise.} \end{cases} . \quad (5.4b)$$

The rule ensures that the average weight follows the product of input voltages probabilistically, except for the "hard wall" restrictions on the weight range. Of course this method requires an external system that reads out Δw_{ij}, performs the addition of the increments, and writes the aggregate weight back into the secondary arrays. However, this operation is much simpler than the multiplication (which is done in parallel by the multiplication arrays), so that in-situ training will still be much faster than software training of a precursor network.

There is still the dilemma between the necessary random exploration and the detrimental effect of noisy updates to the discreet training. This can also be resolved by exactly the same two-unit weight method used to eliminate the relaxation term. To do this, we simply fix the input pattern, and keep updating the multiplication array for $m \gg 1$ iterations without the reset of Δw_{ij} before reading out the finite weight change, and adding it to the weight-representing array. (At that point, the switches in the multiplication arrays are reset to neutral.) Since the switching probability change is typically very small, the relaxation effects in the first array still may be ignored during all m iterations. Formally, this approach may still be described by Eq. (5.4), but with $dw = \sum_{t-1}^{m} \Delta w_{ij}^{t}/m$, where Δw_{ij}^{t} is the weight change obtained at t_{th}

iteration.

In this way, the multiplication arrays are used as effective noise filters; so that the fluctuations can be effectively suppressed (as long as m is sufficiently large), while keeping the stochastic exploration of the phase space necessary for reinforcement learning intact. Of course, the training time is proportional to m, but in CMOL CrossNets with realistic parameters (Sec. 1.6) the noise bandwidth is of the order of 109 Hertz [40, 45], so that even with $m \sim 100$ (see Fig. 5.3 below) one weight increment may still take much less than a microsecond. Note that this method differs from merely using a small learning rate, in that it allows us to calculate the average weight change under fixed inputs and weights, and therefore to implement the exact definition of Δw_{ij} (A.5) for a given input pattern.

There are two possible ways of implementing random fluctuations required for the reinforcement learning rules. One is to use the natural shot and thermal noise of the synapse current to implement Eq. (4.17). The second one is to use two sets of latching switches: one responsible for storing average weights $\langle w_{ij} \rangle$, and the other one as the physical source of fluctuations (Sec. 4.4). The former group is adjusted using the discrete learning rule Eq. (5.4), with short voltage pulses to ensure small switching probabilities. The second set of elementary synapses is agitated with, for example, periodic square waveforms of alternating polarity, with the total time period $T = T_+ + T_-$ (which satisfies condition $T \ll 1$), for an extended period of time mT. The integer m controls the fluctuation variance, while ratio T_+/T_- determines the bias. (For $T_+ = T_-$, the synaptic weight produced by the agitated switches fluctuates with zero mean.) In either case, the random component is equivalent to a Gaussian fluctuation added to x_i (4.17); and that is what we have used in our simulations presented below.

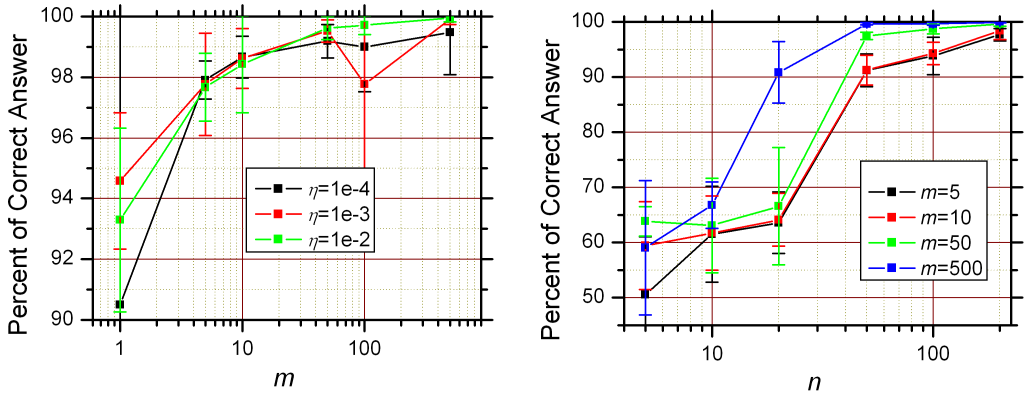
Fig. 5.3 shows a typical dependence of the generalization performance of CrossNets, trained using the in-situ version of rule A2, on parameters m and n, while the last line of Table 5.1 present the results (for relatively large n and m) for the MONK's problems. One can see that in-situ training produces results quite comparable with, and in some cases even better than those for the corresponding continuous weight (e.g., software-implemented) networks.

5.3 Conclusion and Discussion

In the mixed-signal neuromorphic networks - "CrossNets" - implemented using hybrid CMOS/nanoelectronic circuits, the synaptic weights are naturally random. Earlier it was shown that this situation invites new rules of global reinforcement training, and that for networks with stochastic synapses these

Table 5.1: Optimized Generalization Performance for in situ training on the MONK's Problems

Algorithm	Problem 1	Problem 2	Problem 3
$A_{r\text{-}p}$	99.4±0.5 at $\eta = 0.6$, $\lambda = 0.035$	99.8±0.3 at $\eta = 0.7$, $\lambda = 0.025$	96.8±0.0 at $\eta = 0.07$, $\lambda = 0.001$
Software A2	96.3±4.0 at $\eta = 0.08$, $\sigma(0) = 1.8$, $\lambda = 0.005$, $\alpha = .4$	99.7±0.5 at $\eta = 0.1$, $\sigma(0) = 1$, $\lambda = 0.003$, $\alpha = .2$	96.8±0.0 at $\eta = 0.007$, $\sigma = 1$, $\lambda = 0.004$, $\alpha = 1$
Hardware A2	94±2 at $\eta = 0.01$, $m = 50$, $n = 50$	99.95±0.14 at $\eta = 0.01$, $m = 500$, $n = 200$	96.9±0.3 at $\eta = 10^{-5}$, $m = 500$, $n = 100$



(a) $n = 200$.

(b) $\eta = 0.01$.

Figure 5.3: Generalization performance of in-situ training with A2 for the second of the MONK's Problem, as a function of m and n. The parameters not shown in figures are the same as optimized for the continuous weights [64, 65]. The results are averaged over 8 independent experiments; the error bar represents the standard deviation of these samples.

rules may provide generalization performance on a par with that of networks with random somas ("Boltzmann machines"), trained using learning rules of the REINFORCE class. (Let us emphasize again that the latter rules are not applicable to the case of random synapse.)

Although the new rules have been suggested with the CMOL CrossNet hardware in mind, some hardware-imposed limitations had to be overcome to make them suitable for internal ("in-situ") training which seems necessary for solving large-scale problems. We have found a way to suppress the unfavorable relaxation and fluctuation effects on synaptic adaptation by using two latching switch arrays for each synapse, and have shown that for at least relatively small classification problems the generalization fidelity provided by two of the new rules (A2 and A3) is quite comparable with that of the software-implemented networks with quasi-continuous synapses. (The speed of such codes, run on any realistic digital computers, is far inferior to the estimated speed of CMOL CrossNets circuits [16, 40, 45].)

Chapter 6

Conclusion and Possible Future Work

6.1 Summary of Main Results

CrossNets, specialized topologies of CMOL chips for massive intercommunications between CMOS cells, can be trained to perform artificial intelligence functions including associative memory, pattern classification, and reinforcement learning [31].

This dissertation has shown that CrossNets can be trained by reinforcement learning rules to perform pattern classification and reinforcement learning tasks. It has also shown that CrossNets has extremely high tolerance to manufacturing defects when they are used as Hopfield networks.

The following is the list of the main results achieved by the author (in the order they appear in this dissertation):

Self-excitation of Recurrent CrossNets (Sec. 1.4.1)

> I have studied the self-excitation of recurrent CrossNets, and calculated the critical gain (i.e., the smallest value of g at which the self-excitation starts) in the case of multi-valued synapses. The theoretical predictions were confirmed by numerical experiments. On the other hand, this calculation has explained the close-to-linear relation between $x_{\rm rms}$ and g for very high gains.

Hebbian Adaptation in Operation Mode (Sec. 1.5.2)

> I have shown that a similar Hebbian adaptation can be achieved without switching from the "operation" to "training" mode, but instead by simply increasing the load resistance R_L. This may be very useful for

relatively simple learning tasks. Without having to switch the circuit configuration and feed the axonic signals back to dentritic wires, this method would significantly reduce the hardware complexity as well as operation time.

Defect Tolerance of Hopfield CrossNets (Sec. 2.3, 2.4, Refs. [40, 43, 44])

I have studied recurrent CrossNets as Hopfield networks, and calculated defect tolerance of these networks based on the assumption of completely random (uncorrelated) patterns. The calculation was based on methods similar to those used to calculate the storage capacity of the Hopfield networks. I have also carried out a more detailed derivation of network capacity and defect tolerance for the case when the independence assumption is not true (which will be the case for almost any real world application).

Reinforce by Hebbian Adaptation (Sec. 4.1, Ref. [40])

I have shown that by a simple combination of self-excitation and Hebbian adaptation, a recurrent CrossNet can be taught simple classification tasks, using a global reinforcement learning algorithm.

New Reinforcement Rules for CrossNets (Chapter 4, Refs. [64–66])

I have re-derived the REINFORCE approach from a more general point of view (the likelihood ratio method), so that it can be then applied to networks not only with random somas, but also with other sources of randomness. Based on that argument, novel learning rules for networks with randomness from various locations of the network have been proposed. The new rules were tested on both classification and reinforcement tasks and achieved results comparable to those for the known reinforcement learning rules. The new rules, however, are more friendly for CMOL hardware implementation.

In Situ Reinforcement Learning (Chapter 5, Ref. [3])

Difficulties faced in implementing the new learning rules in CrossNets have been addressed, and a solution proposed. When implemented in situ, the new rules can take advantage of both the parallel computation power of the hardware, as well as the intrinsic randomness that comes from nanodevice synapses.

6.2 Future Work

Defect Tolerance of Reinforcement Training

In this dissertation I have shown that defect tolerance of recurrent CrossNets operating in Hopfield mode is extremely high (Sec. 2.3). In Ref. [63] it was shown that the feedforward CrossNets trained as pattern classifiers by either weight import, or in situ training are also very insensitive to both synaptic weight discreteness and nanodevice defects. This kind of robustness comes directly from the parallelism and redundancy nature of neural computing. We expect networks trained by reinforcement learning algorithms to exhibit the same level of resistance to hardware imperfection. However, more experiment results are necessary to confirm these expectations.

Scaling Property of the New Rules

One more important question still open is whether the efficiency of the new rules may be sustained with the growth of network size. Answering this question hinges on finding benchmark problems with variable length L of the input vector, for which the known methods such as $A_{r\text{-}p}$ are insensitive to L. So far we have been unable to find such problems in literature.

In Situ Solution to Cart-Pole

The in situ training solution proposed in Chapter 5 has been so far unsuccessful for the cart-pole balancing problem. This is due to a combination of three factors: the extremely high level of noise in reinforcement learning rules, the discreetness of synapses in hardware, and the "real-time" nature of the control problems. A possible remedy is to average Δw_{ij} for m time steps, and during this time the cart-pole system is free to evolve. This method has been so far unsuccessful, however, even though we have made the training process much faster than the physical process of the cart-pole system in our simulation (see Appendix A for details). This is due to the very much delayed reward for this problem, which further complicates the already difficult learning process. Since meaningful feedback only comes at the time of failure, much time may be wasted averaging the completely random weight changes at the early stage of learning. Probably this difficulty can be resolved by changing the relative speed of the cart-pole system to the CrossNet system during the learning process: a high learning speed at the beginning, and a reduced speed only at the last stage of training.

Possible Connections with Neurobiology

The reinforcement learning is more biologically plausible than the supervised learning. Both assumptions of the noise source and the global rewards/penalties are believed to be likely to exist in the biological brain. In fact, it is more reasonable to assume that the noise comes from the synaptic transmitter release process, rather than in the spike firing mechanism (see the discussion about "reward-based learning" in Ref. [125]). Note that this is exactly what we are proposing for the source of randomness for the new learning rules discussed in Chapter 4. We should explore the possibility of bridging the large scale CrossNet experiment with neurobiology research.

Noise Reduction Techniques

A mechanism for controlling the plasticity of individual neurons is proposed in Ref. [125] to reduce the fluctuations of "non-learning" neurons. This effectively suppress the unnecessary noises while try to retain only the "useful" fluctuations. The proposal should be further studied as it may provide means to significantly reduce the noise of the learning rules that is causing major difficulties in in situ training.

Appendix A

Noises in the Learning Rules

To better understand the difficulty in in situ training with A2, let us compare A2 with BP on the third of the Monk's problem, for which we have achieved 98% of correct classification in the case of continuous weights. The network is (as usual) a 17-3-1 MLP with the following relations between pre- and post-synaptic signals:

$$x_i = \sum_j w_{ij} y_j, \tag{A.1}$$

$$h_i = \frac{G}{\sqrt{N_{m-1}}} x_i, \tag{A.2}$$

$$y_i = \tanh(h_i). \tag{A.3}$$

In the case of discrete weights, we use the in situ learning rule Eq. (5.4). According to A2 (for the case of additive noise in x_i),

$$x_i = \langle x_i \rangle + \delta x_i, \tag{A.4}$$

$$\Delta w_{ij} = \eta \left[r(x_i - \langle x_i \rangle) + \lambda(1-r)(-x_i - \langle x_i \rangle) \right] y_j. \tag{A.5}$$

And as usual, a global noise is quenched according to the sliding average reward:

$$r_a(t) = 0.99 r_a(t-1) + 0.01 r(t), \tag{A.6}$$

$$\sigma(t) = \sigma(0)[1 - r_a(t)]^\alpha. \tag{A.7}$$

This, without the average mentioned in Sec. 5.2, does not work very well (Fig. A.1(a)), unless the number of quantization levels is unrealistically high (Fig. A.1(b)). Back-prop (BP), on the other hand, can certainly do better (Fig. A.1(c)).

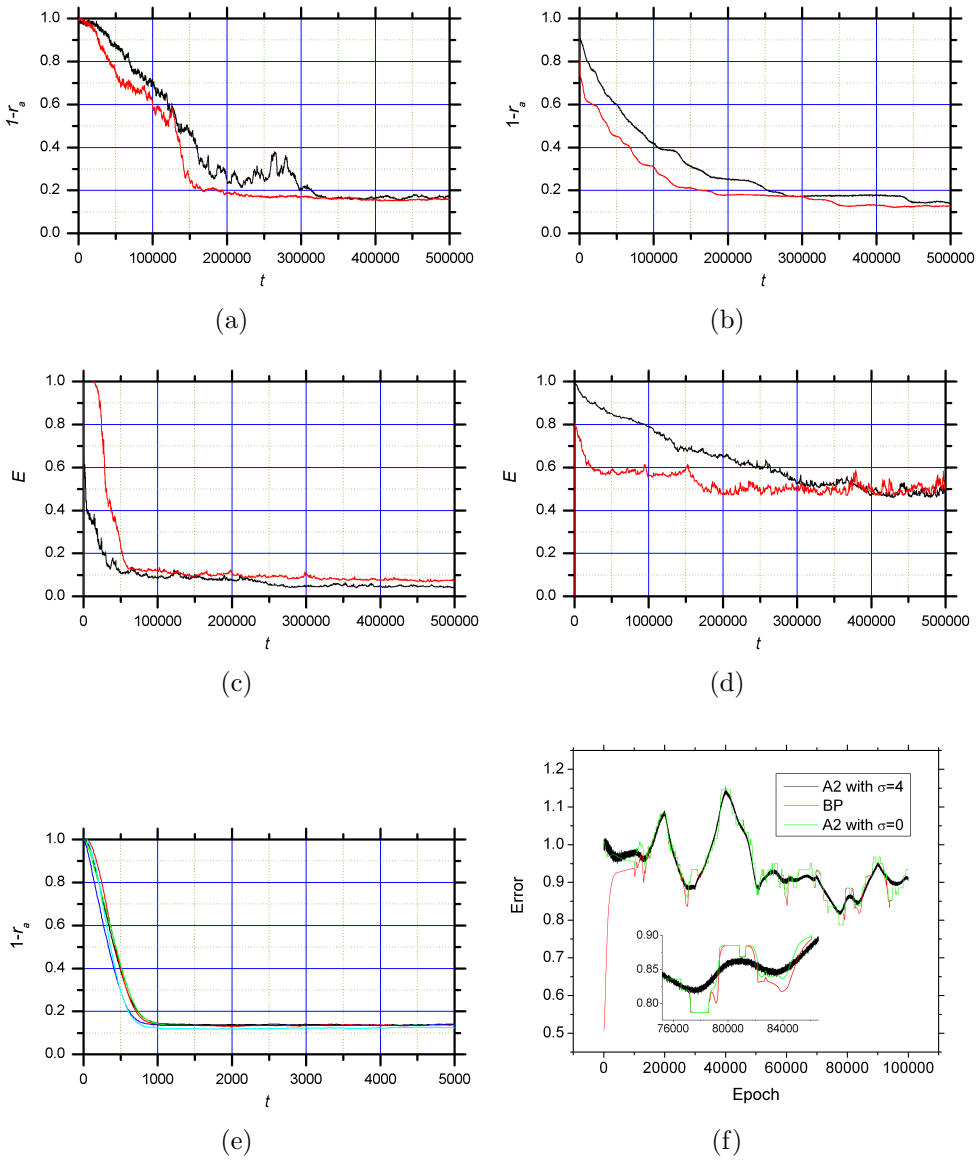


Figure A.1: In situ training without noise filtering. a) In situ training of A2 on the third of the Monk's problem. Parameters: $n = 100$, $G = 0.05$, $\eta = 1.0 \times 10^{-7}$, $\sigma(0) = 4$, $\lambda = 0.005$, $\alpha = 1$. b) A2 with much higher number of quantization levels: $n = 2 \times 10^6$ (the other parameters are the same as in Fig. A.1(a)). c) In situ training with BP: $n = 100$, $G = 0.05$, $\eta = 5.0 \times 10^{-5}$. d) In situ training with BP at small η: $\eta = 1 \times 10^{-5}$ (the other parameters are the same as in Fig. A.1(c)). e) Quasi in situ training with A2. Parameters: $n = 30$, $G = 0.4$, $\eta = 2.5 \times 10^{-5}$, $\sigma(0) = 10$, $\lambda = 0.005$, $\alpha = 1$. f) Visualization of the "cost function" for A2 and BP on the third Monk's problem. Parameters: $\eta = 0.01$, $G = 0.4$.

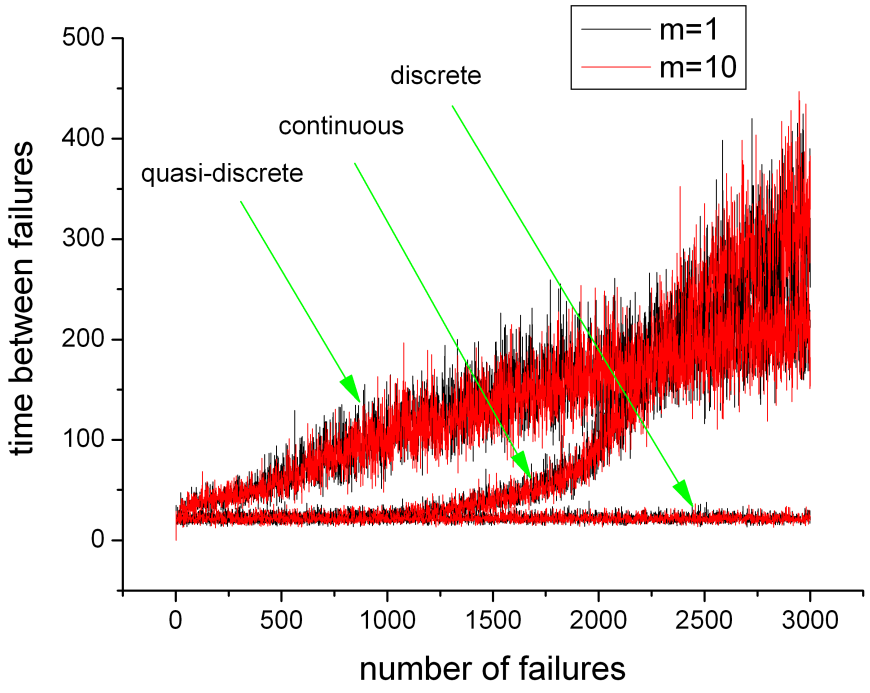


Figure A.2: Learning dynamics of in situ training for the cart-pole task, where m is the number of time steps for accumulating Δw_{ij}, before the weight change is actually applied.

To help us understand the problem better, let us try to visualize the cost function profile in some way. Here I plot the cost function in one dimension by probing the multinational weight space in 10 random directions: $\mathbf{u}^{(k)}$, $k = 1, 2, ..., 10$, where the components $-1 \leq u_{ij}^{(k)} \leq 1$ are drawn from even distribution. During the first 10000 epochs, the weights are updated in the direction $\mathbf{u}^{(1)}$:

$$w_{ij} \longleftarrow w_{ij} + \eta u_{ij}^{(1)}, \tag{A.8}$$

and the they are updated in the direction of $\mathbf{u}^{(2)}$ in the second 10000 epochs, ..., etc.

The cost function for BP is $E = \sum_{p=1}^{P} (T_p - O_p)^2/2P$, where $T_p = \pm 1$ is the target for pattern p and O_p is the output, and the total number of patterns is P. For A2 it is defined as $E = 1 - r(t) = 1 - \sum_{p=1}^{P} T_p O_p/P$. In the case of A2 with noise, the error for an epoch is averaged over 200 runs at fixed weights. The results are shown in Fig. A.1(f).

Whether it is because of the online implementation or the randomness in the learning rule itself, the weight changes are usually noisy:

$$\Delta w_{ij} = \eta \left(\frac{\partial E}{\partial w_{ij}} + \delta \right), \tag{A.9}$$

where δ is some noise with zero mean. This noise can always be subdued by decreasing η. This is because when η is small enough, the partial derivative can be treated as a constant (even in an online implementation) and therefore

$$\langle \Delta w_{ij} \rangle = \eta \frac{\partial E}{\partial w_{ij}} + \eta \langle \delta \rangle = \eta \frac{\partial E}{\partial w_{ij}}$$

Noise may also come from the discretization of the weights. In this case the relative error scales with $1/\sqrt{n}$ and can only be reduced by increasing n. However this noise itself may not be a serious problem. For example, if a continuous variable $0 \leq p_{ij} \leq 1$ is stored and updated by Δw_{ij} (with "hard-wall" restrictions); and at each iteration all the switches are turned on from the off state with probability p_{ij} ($1 - p_{ij}$ for w_{ij}^-), then A2 work fine in this "quasi in situ" training even for small n (Fig. A.1(e)). It is when the true in situ training rule (5.4) is used that it becomes very difficult for A2 to solve the problem completely (Fig. A.1(a)).

Apparently the slowly varying noise in the case of Eqs. (5.4) is more harmful because the error can not be averaged out quickly enough. Another factor that is causing serious problem (especially for A2) is the dilemma between small and large η. Small η helps reduce noise, but from Fig. A.1(f) we can clearly see that a relatively large η is necessary to avoid the numerous local

minima and plateaus in the error profile. To further illustrate this point, Fig A.1(d) shows that even BP stops working at certain small η.

The solution in the Sec. 5.2 failed to work for the cart-pole balance problem. This is because in the case of control tasks, the training happens at real time; and it is physically impossible to fix the state of the cart-pole system (i.e. the input to the actor) while we try different actions. Therefore the average defined in previous section can not be calculated. The alternative method is to average Δw_{ij} for m time steps, and during this time the cart-pole system is free to evolve. This method has been so far unsuccessful (see Fig. A.2), even though we made the training process much faster than the physical process of the cart-pole system in our simulation. Also shown in Fig. A.2 is the result of quasi in situ training (see previous discussions), which works reasonably well. This again shows that it is the combination of the discreteness in the weights and the noise in the learning rule that is causing the problems in A2.

Appendix B

Generalization Comparison of Back-Propagation (BP) and Reinforcement

B.1 Experiment Setup

The experiments were done on fully connected there-layer MLPs. Units in the input layer have fixed output values and hidden layers and all the other units have tanh activation function.

$$x_i = \sum_j w_{ij} y_j \tag{B.1}$$

$$y_i = \tanh(g_i x_i) \tag{B.2}$$

For the reinforcement algorithm, the networks are composed of discrete stochastic units, so

$$p_i = \frac{\tanh(g_i x_i) + 1}{2} \tag{B.3}$$

$$y_i = \begin{cases} +1, & \text{with probability } p_i; \\ -1, & \text{with probability } 1 - p_i. \end{cases} \tag{B.4}$$

The learning rule for BP is the usual delta rule

$$\Delta w_{ij} = \eta \delta_i y_j \tag{B.5}$$

The error for the output node is

$$\delta_o^\mu = g'(h_o)(T^\mu - y_o^\mu) \tag{B.6}$$

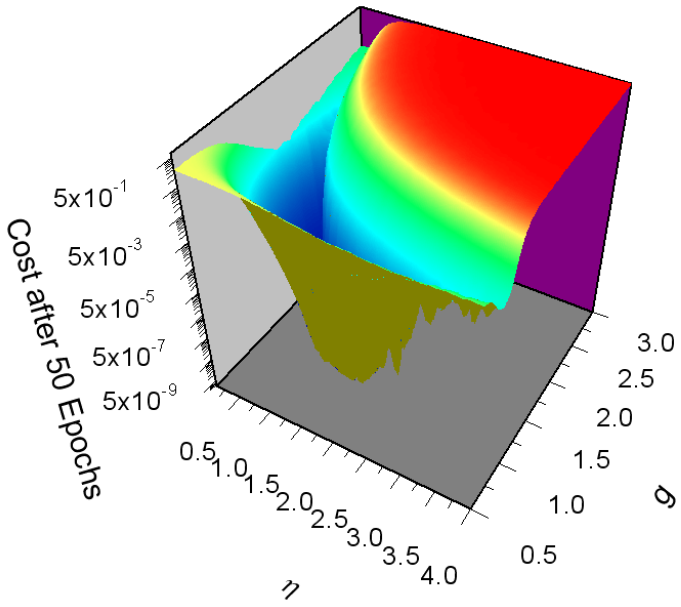


Figure B.1: Cost function after 50 epochs of training. For each specific value of (η,g), the system was trained 200 times and the result was averaged. The raw data was then smoothed and plotted.

where T_μ is the target for current Input I^μ, and y_o^μ is the corresponding output of the output node. Error for the other nodes are calculated by back-propagating the error signal. Bias units are usually added to the input layer and hidden layer. They are the same as the other units except that they have constant output $y_b = +1$.

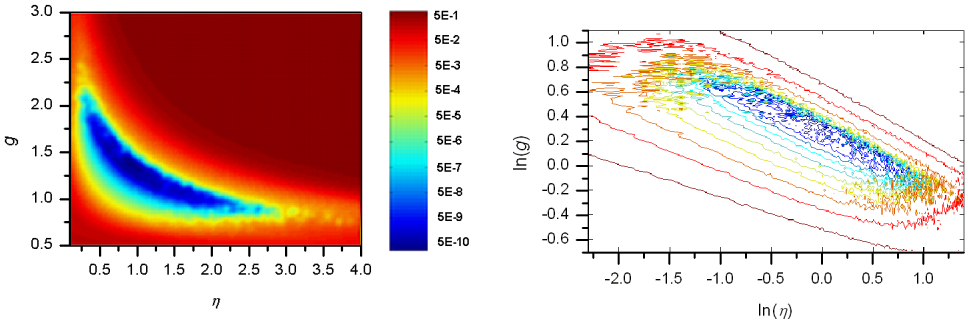
B.2 BP Training Parameters

B.2.1 η and g

An MLP of 2 input units, 10 hidden units and 1 output unit is trained with BP to implement the XOR function. 2 bias units are added to the input layer and 4 bias units are added to hidden layer. All bias units have constant output +1. The inputs and targets are, respectively

$$I = \left\{ \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ +1 \end{pmatrix}, \begin{pmatrix} +1 \\ -1 \end{pmatrix}, \begin{pmatrix} +1 \\ +1 \end{pmatrix} \right\} \tag{B.7}$$

$$T = \left\{ +0.6, \quad -0.6, \quad -0.6, \quad +0.6 \right\} \tag{B.8}$$



(a) (b)

Figure B.2: (a): 2D color map representation of cost function after 50 epochs. Top view of the previous figure, with logarithmic color map. (b): Contour lines with both η and g in ln scale.

Gains for hidden and output units are, respectively

$$g_h = g/\sqrt{2} \tag{B.9}$$

$$g_o = g/\sqrt{10} \tag{B.10}$$

The training results under different g and learning rate η are shown in the Fig B.1 and Fig B.2.

From Fig B.2, (b) we can see that roughly the cost function will be a constant if η and g satisfy the following relation:

$$g^2\eta = A \tag{B.11}$$

where A is some constant.

B.2.2 Bias Amplitude

Based on the results from the previous experiment, the following parameters were chosen for the following experiments: $\eta = 0.95$, $g = 1.32$. In the following I have added one bias unit to both input layer and output layer, but control the amplitude of the bias through parameter b. So bias nodes have the following constant output.

$$y_b^l = b\sqrt{n_l} \tag{B.12}$$

where n_l is the number of normal units in layer l. The result of training is plotted against different b in Fig B.3

For hardware implementation, it is better to increase the number of bias

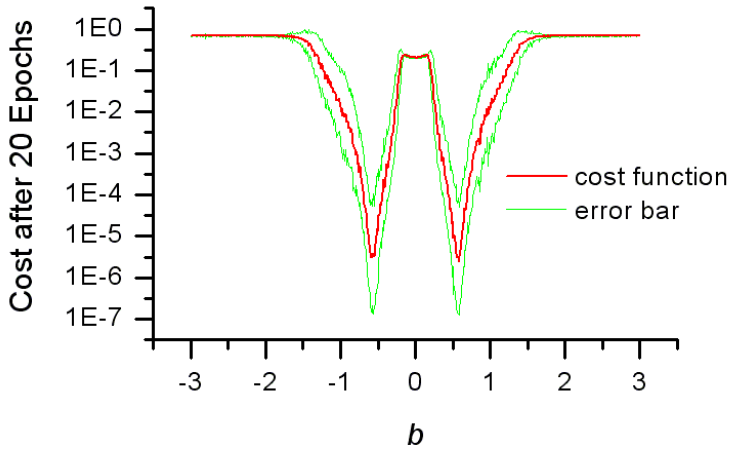


Figure B.3: Cost function after 20 Epochs of training, under different bias amplitude. The result was an average of 200 training. Mean and variance were calculated after applying the $\log_{10}$ function.

units, instead of increasing the output amplitude. For example, I can set the number of bias nodes for layer l to be

$$n_b^l = [b\sqrt{n_l}] \tag{B.13}$$

where $0 \leq b \leq 3$, and $[x]$ is the closest integer to x. All the bias units will have constant output $+1$.

Note that all the weights connecting the bias units in a certain layer to the same regular unit in the next layer change exactly the same way during training, although the initial value of those weights are independent random numbers. Therefore the effect of this bias configuration is similar to that in the previous experiment. But the disruptive effect at large b is now less severe because the initial value of the effective bias is now smaller(due to cancellation among different random weights). The result is shown in Fig B.4.

B.2.3 Target Amplitude

Based on the results from previous experiment, the following experiment was done with 1 bias unit for input layer, and 3 bias units for hidden layer. All of the bias units have constant output $+1$.

As we saw in earlier experiments, how one choose target also affects training speed. If the target is too close to the saturation point of the output node, then it would be very difficult to decrease cost function to very small value.

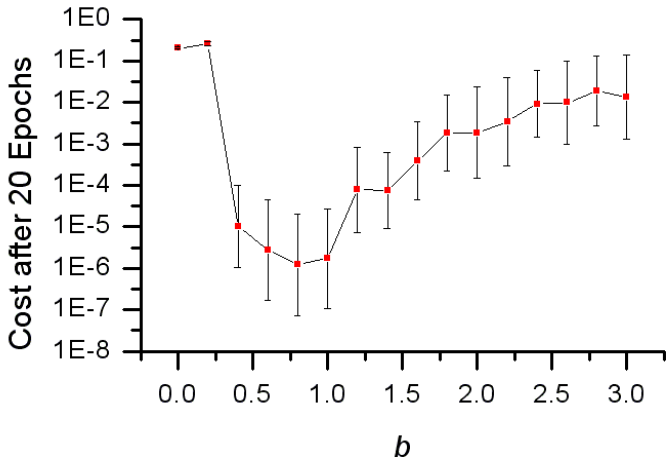


Figure B.4: Cost function after 20 Epochs of training, under different number of bias units. The relation between the number of bias units and parameter b is shown in Eq. (B.13). The result was an average of 200 training. Mean and variance were calculated after applying the $\log_{10}$ function.

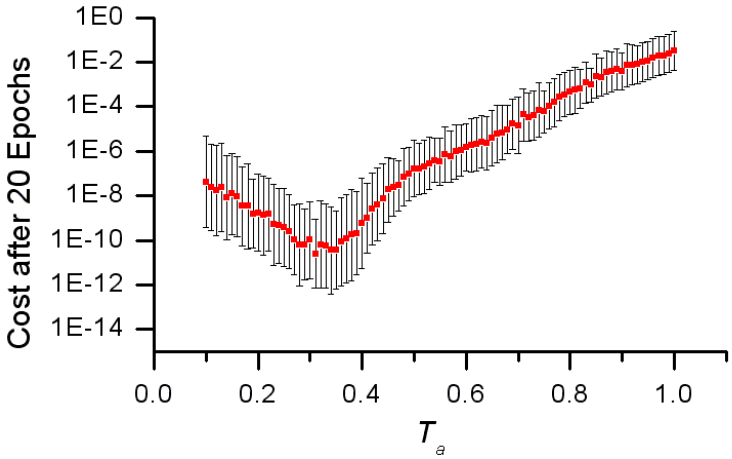


Figure B.5: Training with different target amplitude. The result was an average of 200 training. Mean and variance were calculated after applying the $\log_{10}$ function.

In this section I experiment on different target amplitude T_a, so

$$T = T_a \begin{pmatrix} +1 & -1 & -1 & +1 \end{pmatrix} \tag{B.14}$$

where $0 < T_a \leq 1$. Fig B.5 shows the result of 20 epochs of training with respect to different T_a.

B.3 Reinforcement Training Parameters

The system composed of stochastic units described by Eq. (B.4) was trained with the following learning rule:

$$\Delta w_{ij} = \eta g_i [r(y_i - < y_i >) + \lambda \frac{1-r}{2}(-y_i - < y_i >)] y_j \tag{B.15}$$

to implement parity function with 8 input bits. 190 patterns were used as training set. There were 8 input units, 20 hidden units, and 1 output unit. 2 bias units were added into input layer and 3 to the hidden layer.

For a give input pattern I^μ(μ is the index to the patterns in the training set), and the corresponding output generated by the system V_o^μ, the reward is

$$r^\mu = T^\mu y_o^\mu \tag{B.16}$$

where T^μ is the corresponding target. Eq. (B.16) was used as reward in learning rule Eq. (B.15).

The performance of the network was measured by the average reward. First, the average reward for a complete epoch was

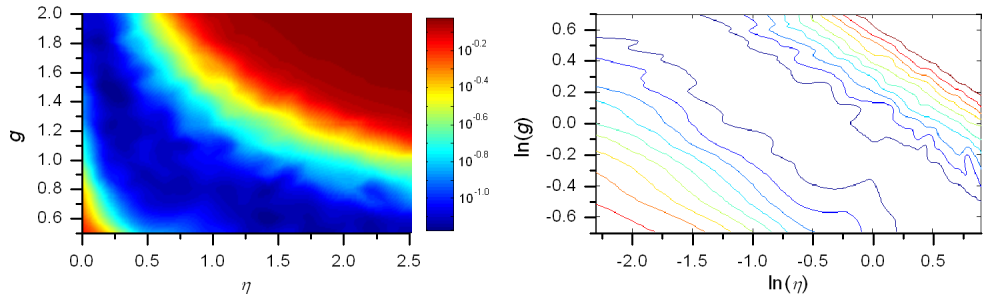
$$r_t = \frac{1}{N_T} \sum_{\mu=1}^{N_T} r^\mu \tag{B.17}$$

where N_T is the total number of patterns in the training set. Time t is in the unit of epochs. Then the average reward at t(averaged over time) was calculated as follows

$$< r >_t = < r >_{t-1} (1-\gamma) + r_t \gamma \tag{B.18}$$

where $\gamma = 0.01$.

Fig B.6 shows $C_r = 1 - < r >_t$ after 800 epochs of training, averaged over 200 experiments.



(a) (b)

Figure B.6: (a): 2D color map representation of equivalent cost C_r after 800 epochs of reinforcement training. Color map is in log scale. (b): Contour lines with both η and g in ln scale.

B.4 Generalization Performance of BP and Reinforcement

B.4.1 Generalization Performance of BP Training

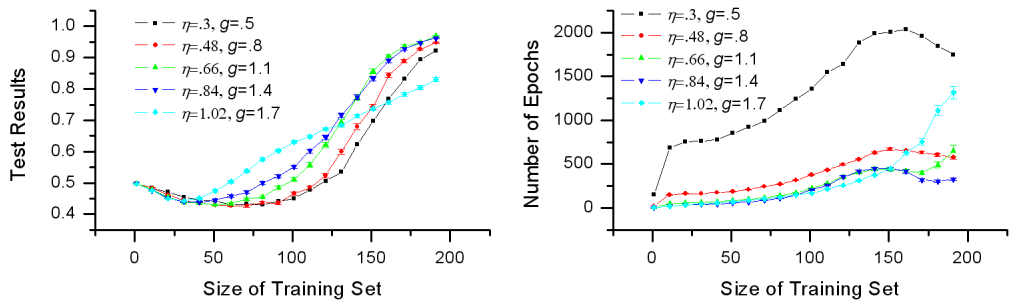
The generalization performance of BP algorithm on parity function with 8 input bits was tested. The network was composed of 8 input units, 20 hidden units and 1 output unit. 2 bias units were added to the input layer and 3 to the hidden layer.

In the experiment, a number of patterns were randomly chosen from the pool of all possible 2^8 input-output pairs as training set. The rest of the possible patterns were used as test set. For each corresponding size of training set, experiment were repeated 200 times, each time with a new training set randomly drawn from the pool. The results were averaged. The cost function is

$$C = \frac{1}{2N_T} \sum_{\mu=1}^{N_T} (T^\mu - y_o^\mu)^2 \tag{B.19}$$

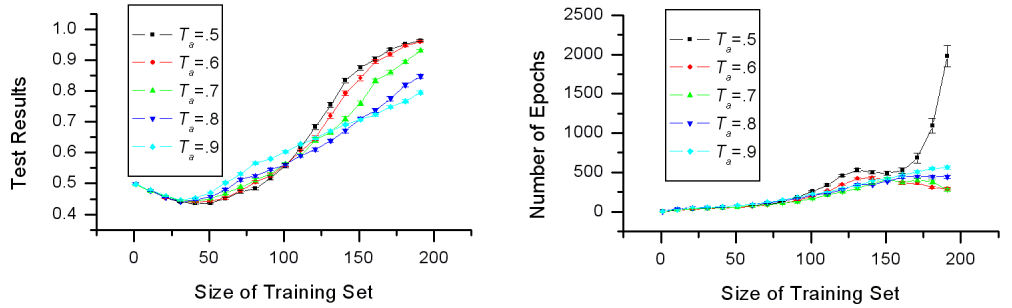
When C reaches the goal of 0.001 the training stops and the trained system was tested on the test set(the test set include all those patterns the system has not seen). If the goal is not met after a maximum of 5000 epochs, training fails and the result is discarded.

Fig B.7 shows the test results for different η and g. And Fig B.8 shows the test results under different T_a.



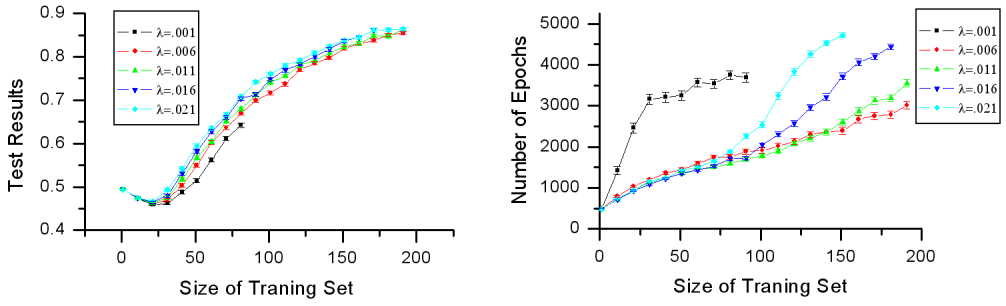
(a) (b)

Figure B.7: (a): Percent of correct mapping on test set after BP training. (b): Average number of epochs needed for C to reach the goal, 5000 epochs was counted for failed training. In (a) and (b) $T_a = .6$.



(a) (b)

Figure B.8: (a): Percent of correct mapping on test set after BP training. (b): Average number of epochs needed for C to reach the goal, 5000 epochs was counted for failed training. In (a) and (b) $\eta = .84$ and $g = 1.4$.



(a) (b)

Figure B.9: (a): Percent of correct mapping on test set after reinforcement training. (b): Average number of epochs needed for C_r to reach the goal, 8000 epochs was counted for failed training.

B.4.2 Generalization Performance of Reinforcement Training

The same system described in the previous section was trained by the reinforcement algorithm (B.15). Training and testing were repeated 100 times for each size of training set. Because the system is stochastic, testing was repeated 50 times after each training, and the results were averaged. Therefore for each size of training set, a total of 5000 test results were averaged.

The parameters are $g_h = .8/\sqrt{8}$, $g_o = .8/\sqrt{20}$, and $\eta = 0.8$. Training succeeds when $C_r = 1 - < r >_t$ reaches the goal of 0.001 and fails after a maximum of 8000 epochs. The percent of correct mapping was shown on Fig B.9, under different λ.

B.4.3 Comparison

The best results from BP and reinforcement training are plotted in Fig B.10.

Conclusion In order to compare the generalization performance of BP and reinforcement, both algorithms have been optimized with respect to training parameters. We have used the parity function problem for this comparison. And we have used $A_{r\text{-}p}$ learning rule for reinforcement training. According to the final results (Fig. B.10), the reinforcement algorithm generalizes much better than BP at small size of training set. Therefore the reinforcement algorithm seemed to be able to "understand" the underlining rule (parity) faster than BP. BP only performers better at large size of training set (close to 55% of all possible input-output pairs),

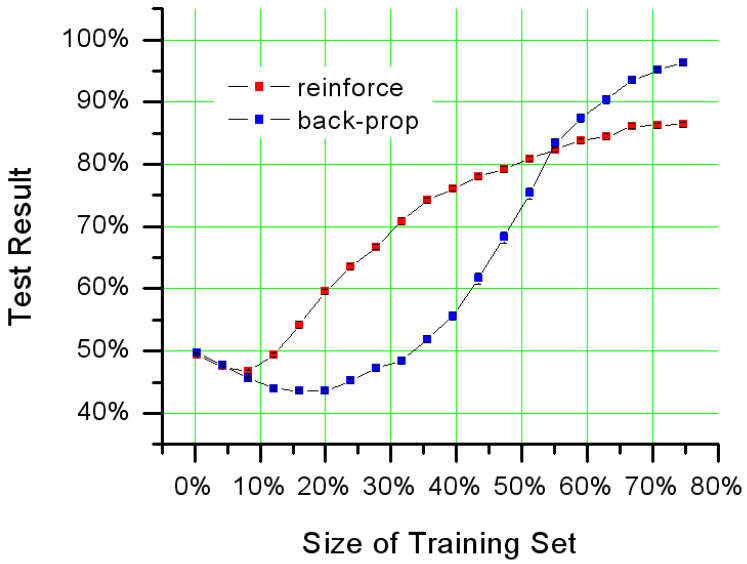


Figure B.10: Generalization performance of reinforcement and BP training. The size of training set is in the form of percentage relative to maximum number of patterns.

due to its advantage of precision. But these are preliminary results. It may still be possible to further optimize the generalization performance of both algorithms.

Bibliography

[1] John Hertz, Anders S. Krogh, and Richard G. Palmer. *Introduction to the theory of neural computation*. Perseus, Cambridge, MA, 1991.

[2] W. S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[3] Ö Türel. *Devices and Circuits for Nanoelectronic Implementation of Artificial Neural Networks*. Doctor of philosophy, Physics and Astronomy, Stony Brook University, June 2007.

[4] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, USA 79:2554–2558, 1982.

[5] Terrence J. Sejnowski and Charles R. Rosenberg. Nettalk: a parallel network that learns to read aloud. *Neurocomputing: foundations of research*, pages 661–672, 1988.

[6] Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[7] Y. Le Cun *et al.* Handwritten digit recognition with a back-propogation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, Denver, CO, 1990. Morgan Kaufman.

[8] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and H. S. P. Wong. Device scaling limits of si MOSFETs and their application dependencies. In *Proc. IEEE*, volume 89, pages 259–288, 2001.

[9] K. K. Likharev. Electronics below 10 nm. *Nano and Giga Challenges in Microelectronics*, pages 27–68, 2003.

[10] International technology roadmap for semiconductors, 2003.

[11] K. K. Likharev. Single-electron devices and their applications. In *Proc. IEEE*, volume 87, pages 606–632, 1999.

[12] M. R. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach, and M. M. Ziegler. Molecular electronics: From devices and interconnect to circuits and architecture. In *Proc. IEEE*, volume 91, pages 1940–1957, 2003.

[13] A. DeHon and K. K. Likharev. Hybrid CMOS/nanoelectronic digital circuits: devices, architectures, and design automation. In *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 375–382, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7803-9254-X.

[14] S. C. Goldstein. The impact of the nanoscale on computing systems. In *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 655–661, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7803-9254-X.

[15] Philip J. Kuekes, Gregory S. Snider, and R. Stanley Williams. Crossbar nanocomputers. *Scientific American*, 293(5):72 – 80, 2005. ISSN 00368733.

[16] K. K. Likharev and D. B. Strukov. *CMOL: Devices, circuits, and architectures*, chapter 16. Berlin: Springer, 2005.

[17] K. K. Likharev. CMOL: Freeing advanced lithography from the alignment accuracy burden. *accepted for publication in J. Vac. Sci. Technol. B*, 25, Nov./Dec. 2007.

[18] K. K. Likharev and D. B. Strukov. Prospects for the development of digital cmol circuits. In *accepted for presentation at NanoArch07*, Santa Clara, Oct. 2007.

[19] K. L. Jensen. Field emitter arrays for plasma and microwave source applications. *Phys. Plasmas*, 6(5):2241–2253, 1999.

[20] S. Fölling, Ö. Türel, and K. K. Likharev. Single-electron latching switches as nanoscale synapses. In *Proceedings of the 2001 International Joint Conference on Neural Networks*, pages 216–221, Mount Royal, NY, 2001. International Neural Network Society.

[21] S. Zankovych, T. Hoffmann, J. Seekamp, J. U. Bruch, and C. M. S. Torres. Nanoimprint lithography: Challenges and prospects. *Nanotechnology*, 12(2):91–95, 2001.

[22] S. R. J. Brueck. There are no fundamental limits to optical lithography. In *International Trends in Applied Optics*, pages 85–109, Bellingham, WA, 2002. SPIE Press.

[23] M. Bender *et al.* Status and prospects of UV-nanoimprint technology. *Microelectronic Engineering*, 83(4-9):827–830, 2006.

[24] G.-Y. Jung *et al.* Circuit fabrication at 17 nm half-pitch by nanoimprint lithography. *Nano Letters*, 6(3):351–354, 2006. ISSN 1530-6984.

[25] J. E. Green *et al.* A 160-kilobit molecular electronic memory patterned at 10^{11} bits per square centimetre. *Nature*, 445:414–417, January 2007. doi: 10.1038/nature05462.

[26] An Chen *et al.* Non-volatile resistive switching for advanced memory applications. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pages 746–749, Dec. 2005.

[27] Y. C. Chen *et al.* Ultra-thin phase-change bridge memory device using GeSb. *Electron Devices Meeting, 2006. IEDM '06. International*, pages 1–4, 11-13 Dec. 2006. doi: 10.1109/IEDM.2006.346910.

[28] D. B. Strukov and K. K. Likharev. Defect-tolerant architectures for nanoelectronic crossbar memories. *Journal of Nanoscience and Nanotechnology*, 7(1):151–167, Jan. 2007.

[29] D. B. Strukov and K. K. Likharev. Prospects for terabit-scale nanoelectronic memories. *Nanotechnology*, 16:137–148, 2005.

[30] D. B. Strukov and K. K. Likharev. CMOL FPGA: A cell-based, reconfigurable architecture for hybrid digital circuits using two-terminal nanodevices. *Nanotechnology*, 16:888–900, 2005.

[31] X. Ma, D. B. Strukov, J. H. Lee, and K. K. Likharev. Afterlife for silicon: CMOL circuit architectures. In *Proc. of the 5th IEEE Conf. on Nanotechnology*, volume 1, pages 175–178, 2005.

[32] Chris M. Bishop. Neural networks and their applications. *Review of Scientific Instruments*, 65(6):1803–1832, 1994. doi: 10.1063/1.1144830.

[33] L. Reyneri. Implementation issues of neuro-fuzzy hardware: Going toward hw/sw codesign. *IEEE Transactions on Neural Networks*, 14:176, 2003.

[34] F. M. Dias, A. Antunes, and M. Mota. Artificial neural networks: a review of commercial hardware. *Engineering Applications of Artificial Intelligence*, 17:945, 2004.

[35] J. Greer, A. Korkin, and J. Labanowski. *Nano and Giga Challenges in Microelectronics*, chapter 2, page 2768. Elsevier, Amsterdam, Netherlands, 2003.

[36] S. Bandyopadhyay and V. Roychowdhury. Computational paradigms in nanoelectronics: Quantum coupled single electron logic and neuromorphic networks. *Japanese Journal of Applied Physics Part 1 - Regular Papers Short Notes and Review Papers*, 35(6A):3350–3362, June 1996.

[37] N. Kouklin P. F. Williams S. Bandyopadhyay, L. Menon and N. J. Ianno. Self-assembled networks with neural computing attributes. *Smart Materials and Structures*, 11(5):761–766, 2002.

[38] Alexandre Schmid and Yusuf Leblebici. Robust circuit and system design methodologies for nanometer-scale devices and single-electron transistors. *IEEE Trans. Very Large Scale Integr. Syst.*, 12(11):1156–1166, 2004.

[39] Ö. Türel and K. K. Likharev. Crossnets: possible neuromorphic networks based on nanoscale components. *Int. J. of Circuit Theory and Appl.*, 31: 37–52, 2003.

[40] Ö. Türel, J. H. Lee, X. L. Ma, and K. K. Likharev. Neuromorphic architectures for nanoelectronic circuits. *Int. J. of Circuit Theory and Applications*, 32(5):277–302, 2004.

[41] Ö. Türel, I. Muckra, and K. K. Likharev. Possible nanoelectronic implementation of neuromorphic networks. In *International Joint Conference on Neural Networks*, pages 365–370. International Neural Network SocietySociety, Mount Royal, New York., 2003.

[42] K. K. Likharev, A. Mayr, I. Muckra, and Ö. Türel. Crossnets: High-performance neuromorphic architectures for cmol circuits. In J. Reimers *et al.*, editor, *Molecular Electronics III*, volume 1006, pages 146–163. Ann. New York Acad. Sci., 2003.

[43] Ö. Türel, J. H. Lee, X. L. Ma, and K. K. Likharev. Nanoelectronic neuromorphic networks (crossnets): New results. In *International Joint Conference on Neural Networks,*, pages 389–394, Budapest, Hungary, 2004.

[44] Ö. Türel, J. H. Lee, X. L. Ma, and K. K. Likharev. Architectures for nanoelectronic implementation of artificial neural networks: New results. *Neurocomputing*, 64:271–283, 2005.

[45] J. H. Lee, X. Ma, and K. K. Likharev. CMOL crossnets: Possible neuromorphic nanoelectronic circuits. In Y. Weiss and B. Sch editors, *Advances in Neural Information Processing Systems*, 2006.

[46] L. Fausett. *Fundamentals of Neural Networks*. Prentice-Hall, Upper Saddle River, NJ, 1994.

[47] M. H. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, MA, 1995.

[48] S. Haykin. *Neural Networks*. Prentice-Hall, Upper Saddle River, NJ, 1999.

[49] H. Park *et al.* Nanomechanical oscillations in a single-C_{60}. *Nature*, 407 (6800):57–60, 2000.

[50] S. P. Gubin *et al.* Molecular clusters as building blocks for nanoelectronics: the first demonstration of a cluster single-electron tunneling transistor at room temperature. *Nanotechnology*, 13(2):185–194, 2002.

[51] N. B. Zhitenev, H. Meng, and Z. Bao. Conductance of small molecular junctions. *Phys. Rev. Lett.*, 88(22):226801, May 2002. doi: 10.1103/PhysRevLett.88.226801.

[52] J. Park *et al.* Coulomb blockade and the kondo effect in single-atom transistors. *Nature*, 417(6890):722–725, 2002.

[53] S. Kubatkin *et al.* Single-electron transistor of a single organic molecule with access to several redox states. *Nature*, 425(6959):698–701, 2003.

[54] J. R. Heath, P. K. Kuekes, G. S. Snider, and R. S. Williams. A defect-tolerant computer architecture: opportunities for nanotechnology. *Science*, 280(5370):1716–1721, 1998.

[55] W. D. Brown and J. E. Brewer, editors. *Nonvolatile Semiconductor Memory Technology*. IEEE Press, Piscataway, NJ, 1998.

[56] P. J. Kuekes, D. R. Stewart, and R. S. Williams. The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits. *Journal of Applied Physics*, 97:4301–+, February 2005.

[57] C. P. Collier *et al.* Electronically Configurable Molecular-Based Logic Gates. *Science*, 285(5426):391–394, 1999. doi: 10.1126/science.285.5426.391.

[58] Charles P. Collier *et al.* A [2]Catenane-Based Solid State Electronically Reconfigurable Switch. *Science*, 289(5482):1172–1175, 2000. doi: 10.1126/science.289.5482.1172.

[59] D. Purves *et al. Neuroscience.* Sinauer, Sunderland, MA, 1997.

[60] R. N. Bracewell. *The Fourier Transform and Its Applications*. McGraw-Hill, New York, 2nd edition, 1986.

[61] J. H. Lee and K. K. Likharev. CMOL CrossNets as pattern classifiers. *Lecture Notes on Computer Science*, 3512:446–454, 2005.

[62] J. H. Lee and K. K. Likharev. In situ training of CMOL CrossNets. In *Proc.* WCCI/IJCNN06, pages 5026–5034, July 2006.

[63] J. H. Lee and K. K. Likharev. Defect-tolerant nanoelectronic pattern classifiers. *accepted for publication in Int. J. of Circuit Theory and Applications*, 2007.

[64] X. Ma and K. K. Likharev. Global reinforcement learning in neural networks with stochastic synapses. In *Proc. of the Int. Joint Conf. on Neural Networks 2006*, pages 47–53, July 2006.

[65] X. Ma and K. K. Likharev. Global reinforcement learning in neural networks. *IEEE Tran. on Neural Networks*, 18(2):573–577, March 2007.

[66] X. Ma. An extremely simple reinforcement learning rule for neural networks. *Lecture Notes on Computer Science*, 4491:438–444, 2007.

[67] J. H. Lee. *CMOL CrossNets as Defect-Tolerant Classifiers*. Doctor of philosophy, Physics and Astronomy, Stony Brook University, November 2007.

[68] Daniel J. Amit, Hanoch Gutfreund, and H. Sompolinsky. *Phsycal Review A*, 35:2293–2303, 1987.

[69] D. V. Averin and A. N. Korotkov. Correlated single-electron tunneling via mesoscopic metal particles: Effects of the energy quantization. *Journal of Low Temperature Physics*, 80(3):173–185, 1990.

[70] Hisanao Akima, Shigeo Sato, and Koji Nakajima. Design of single electron circuitry for a stochastic logic neural network. In *Lecture Notes in Computer Science*, volume 3213, pages 1010–1016. Springer Berlin / Heidelberg, 2004.

[71] B. R. Gains. Stochastic computing systems. In J. T. Tou, editor, *Advances in Information Processing Systems*, volume 2, chapter 2, pages 37–172. Plenum Press, New York, 1969.

[72] Y. Kondo and Y Sawada. Functional abilities of a stochastic logic neural network. *IEEE Trans. on Neural Networks*, 3(3):434 – 443, May 1992.

[73] Max van Daalen, Peter Jeavons, and John Shawe-Taylor. A stochastic neural architecture that exploits dynamically reconfigurable FPGAs. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 202–211, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[74] T. J. Hinton, G. E.; Sejnowski. Optimal perceptual inference. 1983.

[75] P. Peretto. Collective properties of neural networks: A statistical physics approach. *Biological Cybernetics*, 50:51–62, 1984.

[76] Roy J. Glauber. Time-dependent statistics of the ising model. *Journal of Mathematical Physics*, 4(2):294–307, 1963. doi: 10.1063/1.1703954.

[77] Daniel J. Amit, Hanoch Gutfreund, and H. Sompolinsky. Spin-glass models of neural networks. *Phys. Rev. A*, 32(2):1007–1018, Aug 1985. doi: 10.1103/PhysRevA.32.1007.

[78] J. L. van Hemmen. *Phys. Rev. Lett.*, 57:913–916, 1986.

[79] D. B. Strukov and K. K. Likharev. A reconfigurable architecture for hybrid CMOS/nanodevice circuits. In *Proceedings of the FPGA06*, pages 131–140, New York, 2006. ACM.

[80] Greg Snider, Philip Kuekes, and R Stanley Williams. CMOS-like logic in defective, nanoscale crossbars. *Nanotechnology*, 15(8):881–891, 2004.

[81] A. DeHon and H. Naeimi. Seven strategies for tolerating highly defective fabrication. *IEEE Design and Test of Computers*, 22(4):306–315, 2005. ISSN 0740-7475.

[82] Wojciech Tarkowski and Maciej Lewenstein. *J. Phys. A: Math. Gen.*, 26:2453–2469, 1993.

[83] E Gardner. *J. Phys. A: Math. Gen.*, 21:257–70, 1988.

[84] E A Dorotheyev, G Rotundo, and B tirozzi. *J. Phys. A: Math. Gen.*, 28:3733–3741, 1995.

[85] Hanoch Gutfreund. *Physical Review A*, 37:570–577, 1988.

[86] Daniel J. Amit, Hanoch Gutfreund, and H. Sompolinsky. *Phsycal Review A*, 35:2293–2303, 1987.

[87] G. WIDROW and M. HOFF. Adaptive switching circuits. Technical report, Institute of Radio Engineers, New York, 1960.

[88] Arthur E. Bryson and Yu-Chi Ho. *Applied Optimal Control*. Blaisdell Publishing Co., New York, 1969.

[89] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[90] D. B. Parker. Learning logic. Tr-47, Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science, Cambridge, MA, 1985.

[91] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[92] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning : An introduction*. MIT Press, Cambridge, MA, 1998.

[93] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[94] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, Massachusetts, 1960.

[95] Eric A. Hansen. Solving POMDPs by searching in policy space. In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 211–219, Madison, WI, 1998. Morgan Kaufmann Publishers.

[96] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[97] Martin L. Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, jul 1978. ISSN 0025-1909.

[98] J. Denker *et al.* Large automatic learning, rule extraction and generalization. *Complex Systems*, 1:887–922, 1987.

[99] Leemon C. Baird. Residual algorithms: Reinforcement learning with function approximation. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37, San Francisco, CA, 1995. Morgan Kaufmann.

[100] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

[101] Gerald Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput.*, 6(2):215–219, 1994. ISSN 0899-7667.

[102] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst., Man, Cybern.*, SMC-13:834–846, 1983.

[103] Satinder Singh and Dimitri Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 974. The MIT Press, 1997.

[104] P. Mazzoni, R. A. Andersen, and M. I. Jordan. A more biologically plausible learning rule for neural networks. *Proceedings of the National Academy of Sciences*, 88(10):4433–4437, May 1991. ISSN 0027-8424.

[105] G. E. Hinton and T. J. Sejnowski. Learning and relearning in boltzmann machines. In D. E. Rumelhart, editor, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 282–317. Cambridge, MA: MIT Press, 1968.

[106] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[107] W. Maass and A. M. Zador. Dynamic stochastic synapses as computational units. *Neural Computation*, 11:903–917, 1999.

[108] H. S. Seung. Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, 40:1063–1073, 2003.

[109] B. Flower and M. Jabri. Summed weight neuron perturbation: An o(n) improvement over weight perturbation. In Morgan Kaufmann, editor, *Advances in Neural Information Processing Systems(NIPS92)*, volume 5, pages 212–219. San Mateo, CA, 1993.

[110] M. Jabri and B. Flower. Weight perturbation - and optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks. *IEEE Trans. Neural Netw.*, 3:154–157, 1992.

[111] K. P. Unnikrishnan and K. P. Venugopal. Alopex: a correlation based learning algorithm for feed-forward and recurrent neural networks. *Neural Computation*, 6:469–490, 1994.

[112] L. Baird. Gradient descent for general reinforcement learning. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 968–974, Cambridge, MA, 1999. MIT Press.

[113] P. Dayan and L. F Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, Cambridge, MA, 2001.

[114] M. M. Aleksandrov, V. I. Sysoyev, and V. V. Shemeneva. Stochastic optimaization. *Engineering Cybernetics*, 5:11–16, 1968.

[115] A. G. Barto and M. I. Jordan. Gradient following without back-propagation in layered networks. In *Proceedings of the First Annual International Conference on Neural Networks*, volume 2, pages 629–636, San Diego, CA, 1987.

[116] P. Dayan and G. E. Hinton. Varieties of helmholtz machine. *Neural Networks*, 9(8):1385–1403, 1996.

[117] K. S. Narendra and M. A. L. Thathatchar. *Learning Automata: An Introduction*. Englewood Cliffs, Prentice Hall NJ, 1989.

[118] J Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[119] S. B. Thrun *et al.* The MONK's problems: A performance comparison of different learning algorithms. Technical Report CS-91-197, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1991.

[120] H. J. Kushner and D. S. Clark. *Stochastic approximation methods for constrained and unconstrained systems*. Springer-Verlag, 1978.

[121] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 950–957. Morgan Kaufmann Publishers, Inc., 1992.

[122] J. S. Albus. A new approach to manipulator conrol: the cerebellar model articulation controller (CMAC). *Trans. of ASME Journal of Dynamic Systems, Measurements, and Control*, 97(3):220–227, 1975.

[123] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, November 1982. doi: 10.1007/BF00275687.

[124] Geoffrey Hinton and Terrence J. Sejnowski, editors. *Unsupervised Learning - Foundations of Neural Computation*. Bradford Company, Scituate, MA, USA, 1999. ISBN 0-262-58168-X.

[125] Paul R. Adams. The thalamocortical algorithm. In Paul Adams and Terry Sejnowski, editors, *Neurocomputational Strategies: From Synapses to Behavior*, Banbury Center Workshop Series. The Banbury Center of Cold Spring Harbor Laboratory, February 1998.